2020

Algebraic approaches for coded caching and distributed computing

Li Tang
Iowa State University

Recommended Citation

Tang, Li, "Algebraic approaches for coded caching and distributed computing" (2020). *Graduate Theses and Dissertations*. 17873.
https://lib.dr.iastate.edu/etd/17873

Algebraic approaches for coded caching and distributed computing

by

Li Tang

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Electrical Engineering

Program of Study Committee:
Aditya Ramamoorthy, Major Professor
Baskar Ganapathysubramanian
Chinmay Hegde
Zhengdao Wang
Sung Yell Song

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this dissertation. The Graduate College will ensure this dissertation is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2020

DEDICATION

I would like to dedicate this thesis to my mother Dan Liu and father Xiaobo Tang. Without their support I would not have been able to complete this work.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Aditya Ramamoorthy for his guidance, patience and support throughout this research and the writing of this thesis. He always encouraged me when I hit the rock bottom in my research and life.

I would also like to thank my committee members for their efforts and contributions to this work: Dr. Baskar Ganapathysubramanian, Dr. Chinmay Hegde, Dr. Zhengdao Wang and Dr. Sung Yell Song.

Finally, I would like to thank all my wonderful friends at Ames, Iowa.

ABSTRACT

This dissertation examines the power of algebraic methods in two areas of modern interest: caching for large scale content distribution and straggler mitigation within distributed computation.

Caching is a popular technique for facilitating large scale content delivery over the Internet. Traditionally, caching operates by storing popular content closer to the end users. Recent work within the domain of information theory demonstrates that allowing coding in the cache and coded transmission from the server (referred to as coded caching) to the end users can allow for significant reductions in the number of bits transmitted from the server to the end users. The first part of this dissertation examines problems within coded caching.

The original formulation of the coded caching problem assumes that the server and the end users are connected via a single shared link. In Chapter 2, we consider a more general topology where there is a layer of relay nodes between the server and the users. We propose novel schemes for a class of such networks that satisfy a so-called resolvability property and demonstrate that the performance of our scheme is strictly better than previously proposed schemes. Moreover, the original coded caching scheme requires that each file hosted in the server be partitioned into a large number (i.e., the subpacketization level) of non-overlapping subfiles. From a practical perspective, this is problematic as it means that prior schemes are only applicable when the size of the files is extremely large. In Chapter 3, we propose a novel coded caching scheme that enjoys a significantly lower subpacketization level than prior schemes, while only suffering a marginal increase in the transmission rate. We demonstrate that several schemes with subpacketization levels that are exponentially smaller than the basic scheme can be obtained.

The second half of this dissertation deals with large scale distributed matrix computations. Distributed matrix multiplication is an important problem, especially in domains such as deep learning of neural networks. It is well recognized that the computation times on distributed clusters

are often dominated by the slowest workers (called stragglers). Recently, techniques from coding theory have found applications in straggler mitigation in the specific context of matrix-matrix and matrix-vector multiplication. The computation can be completed as long as a certain number of workers (called the recovery threshold) complete their assigned tasks.

In Chapter 4, we consider matrix multiplication under the assumption that the absolute values of the matrix entries are sufficiently small. Under this condition, we present a method with a significantly smaller recovery threshold than prior work. Besides, the prior work suffers from serious numerical issues owing to the condition number of the corresponding real Vandermonde-structured recovery matrices; this condition number grows exponentially in the number of workers. In Chapter 5, we present a novel approach that leverages the properties of circulant permutation matrices and rotation matrices for coded matrix computation. In addition to having an optimal recovery threshold, we demonstrate an upper bound on the worst case condition number of our recovery matrices grows polynomially in the number of workers.

CHAPTER 1. INTRODUCTION

Caching and distributed computing are two important technologies in the Big Data era that are widely used in a variety of commercial settings. Broadly speaking both these can be considered as applications of network coding Ahlswede et al. (2000), Li et al. (2003). Network coding that was introduced in Ahlswede et al. (2000) is a generalization of routing where intermediate nodes in a network combine information as it travels through a network. Several problems within network coding have been investigated over the years and various types of network connections have been considered within this domain. These include the original multicast Ho et al. (2006), multiple unicast Dougherty et al. (2005) and Huang et al. (2011), Huang and Ramamoorthy (2013, 2014) and function computation Ramamoorthy and Langberg (2013); Langberg and Ramamoorthy (2009), Rai and Dey (2012) among other topics.

1.1 Coded Caching

Caching is a popular technique for facilitating content delivery over the Internet. It exploits local cache memory that is often available at the end users for reducing transmission rates from the central server. In particular, when the users request files from a central server, the system first attempts to satisfy the user demands in part from the local content. Thus, the overall rate of transmission from the server is reduced which in turn reduces overall network congestion. The work of Maddah-Ali and Niesen (2014b) demonstrated that huge rate savings are possible when coding in the caches and coded transmissions from the server to the users are considered. This problem is referred to as coded caching.

In Maddah-Ali and Niesen (2014b), the scenario considered was as follows. There is a server that contains N files, a collection of K users that are connected to the central server by a single

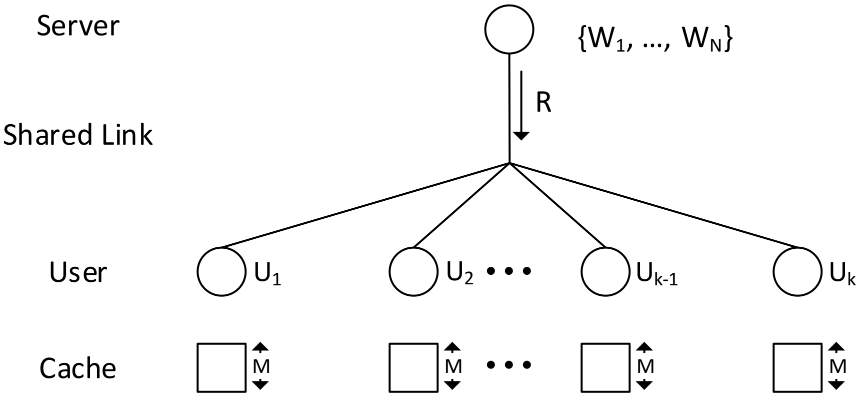

Figure 1.1 Model of coded caching in Maddah-Ali and Niesen (2014b).

shared link. Each user also has a local cache of size M. The focus is on reducing the rate of transmission on the shared link. There are two distinct phases in the coded caching setting.

- *Placement phase:* In the placement phase, the caches of the users are populated. This phase should not depend on the actual user requests, which are assumed to be arbitrary. The placement phase is executed in the off-peak traffic time.

- *Delivery phase:* In delivery phase, server sends some *coded* signal to each user such that each user's demand is satisfied. The delivery phase is always executed in the peak traffic time.

Since the original work of Maddah-Ali and Niesen (2014b), there have been several aspects of coded caching that have been investigated. Maddah-Ali and Niesen (2014a) considered the decentralized coded caching where the placement phase is driven by the users who randomly populate their caches. Information theoretic lower bounds on the transmission rate were considered in Ghasemi and Ramamoorthy (2016), Ghasemi and Ramamoorthy (2017c), Yu et al. (2018). Synchronization issues in this problem setting were investigated in Ghasemi and Ramamoorthy (2017a), Ghasemi and Ramamoorthy (2017b). More recently, techniques inspired by coded caching have been employed for speeding up distributed computing Li et al. (2017), Konstantinidis and Ramamoorthy (2018), Konstantinidis and Ramamoorthy (2019), Konstantinidis and Ramamoorthy (pear).

We explain the idea of coded caching by the following example in Maddah-Ali and Niesen (2014b).

Example 1. Consider the case that $K = N = 2$, $M = 1$, such that there are two files A and B in the server and two users each with cache memory size $M = 1$ file. We describe the coded caching scheme as follows. In the placement phase, file A is split into two non-overlapping subfiles $A = \{A_1, A_2\}$. Similarly B is split into $B = \{B_1, B_2\}$. User 1 caches A_1 and B_1, and thus its cache memory is 1 file. User 2 caches A_2 and B_2, and thus its cache memory is also 1 file. In the delivery phase, suppose for example user 1 requires file A and user 2 requires file B. User 1 needs A_2 and user 2 needs B_1 from server. Server can definitely transmit A_2 and B_1 over the shared link and the transmission rate is 1 file. We call this transmission rate as *uncoded transmission rate*. Coded caching introduces another transmission strategy. The server can simply transmit $A_2 \oplus B_1$, where $\oplus$ denotes bitwise XOR. User 1 already has B_1, thus it can recover A_2 from $A_2 \oplus B_1$. Similarly, user 2 can recover B_1 since it already has A_2. Therefore, the transmission $A_2 \oplus B_1$ can guarantee these two users recover their requests and the transmission rate is half file. We call this transmission rate as *coded transmission rate*. In this example, the coded transmission rate is only half of uncoded transmission rate. Figure 1.2 shows coded transmission rate is always the equivalent of half a file for all the four user requests.

1.1.1 Coded Caching for Networks with the Resolvability Property

Maddah-Ali and Niesen (2014b) demonstrate that by carefully designing the cache in the users, the coded transmission from the central server can significantly save the transmission rate when the server and the end users are connected via a single shared link. In the first part of dissertation, we consider a more general topology where there is a layer of relay nodes between the server and the users. We demonstrate that our proposed scheme outperforms two previous proposed schemes in Ji et al. (2015a). This work has appeared in Tang and Ramamoorthy (2016a) and is discussed in Chapter 2.

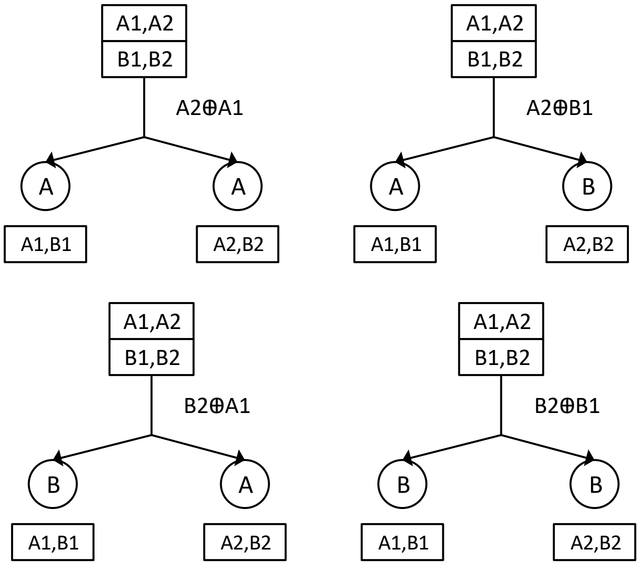

Figure 1.2 Caching strategy for $N = 2$ files and $K = 2$ users with cache size $M = 1$ with all four possible user requests. Each file is split into 2 subfiles. The schemes achieve rate $R = 0.5$.

1.1.2 Coded Caching Schemes with Reduced Subpacketization from Linear Block Codes

Maddah-Ali and Niesen (2014b) proposed a coded caching scheme and showed that compared with conventional caching, coded caching can achieve a much lower transmission rate. However, in the placement phase of Maddah-Ali's scheme, each file is split into a very large number of non-overlapping subfiles of equal size. This is called the *subpacketization level* of the scheme. It means Maddah-Ali's scheme is applicable only in the regime when the underlying file sizes are very large. In the second part of this dissertation, we propose coded caching schemes based on combinatorial structures called resolvable designs. These structures can be obtained in a natural manner from linear block codes whose generator matrices possess certain rank properties. We demonstrate that several schemes with subpacketization levels that are exponentially smaller than the basic scheme can be obtained. The subpacketization level of our scheme is exponentially lower than the memory-sharing within the scheme of Maddah-Ali and Niesen (2014b). This work has appeared in Tang and Ramamoorthy (2018, 2016b, 2017) and is discussed in Chapter 3.

1.2 Distributed computing

The current Big Data era routinely requires the processing of large scale data on massive distributed computing clusters. In these applications, data sets are often so large that they cannot be housed in the memory and/or the disk of any one computer. Thus, the data and the processing is typically distributed across multiple nodes. Distributed computation is thus a necessity rather than a luxury. The widespread usage of such clusters presents several opportunities and advantages over traditional computing paradigms. However, it also presents newer challenges where coding-theoretic ideas have recently had a significant impact. Large scale clusters (which can be heterogeneous in nature) suffer from the problem of stragglers which refer to slow or failed worker nodes in the system. Thus, the overall speed of a computation is typically dominated by the slowest node in the absence of a sophisticated assignment of tasks to the worker nodes.

The conventional approach for tackling stragglers in distributed computation has been to run multiple copies of tasks on various machines, with the hope that at least one copy finishes on time. However, coded computation offers significant benefits for specific classes of problems. We illustrate this by means of a matrix-matrix multiplication example in Yu et al. (2017).

Example 2. Consider a distributed matrix multiplication task of computing $\mathbf{C} = \mathbf{A}^T\mathbf{B}$ using five worker that each store the equivalent of half of the matrices $\mathbf{A}$ and $\mathbf{B}$. We first split the matrices evenly along the column dimensions to obtain $\mathbf{A} = [\mathbf{A}_0, \mathbf{A}_1]$, $\mathbf{B} = [\mathbf{B}_0, \mathbf{B}_1]$. We want to compute the following four matrix products that have shown below.

$$\mathbf{C} = \mathbf{A}^T\mathbf{B} = \begin{bmatrix} \mathbf{A}_0^T\mathbf{B}_0 & \mathbf{A}_0^T\mathbf{B}_1 \\ \mathbf{A}_1^T\mathbf{B}_0 & \mathbf{A}_1^T\mathbf{B}_1 \end{bmatrix}$$

Now we design a computation strategy which is resilient to a single straggler. Let the master node give worker $i \in \{1, 2, 3, 4, 5\}$ the following two submatrices.

$$\hat{\mathbf{A}}_i = \mathbf{A}_0 + i\mathbf{A}_1, \text{ and}$$

$$\hat{\mathbf{B}}_i = \mathbf{B}_0 + i^2\mathbf{B}_1.$$

Each worker compute $\hat{\mathbf{A}}_i^T\hat{\mathbf{B}}_i$. It follows that as soon as *any* four out of the five worker nodes return the results of their computation, the master node can decode and recover $\mathbf{C}$. We discuss this through a representative scenario, where the master receives the computation results from workers 1, 2, 3, and 4. The computation result from worker i is

$$\hat{\mathbf{C}}_i = \hat{\mathbf{A}}_i^T\hat{\mathbf{B}}_i = \mathbf{A}_0^T\mathbf{B}_0 + i\mathbf{A}_1^T\mathbf{B}_0 + i^2\mathbf{A}_0^T\mathbf{B}_1 + i^3\mathbf{A}_1^T\mathbf{B}_1$$

Then we have

$$\begin{bmatrix} \hat{\mathbf{C}}_1 \\ \hat{\mathbf{C}}_2 \\ \hat{\mathbf{C}}_3 \\ \hat{\mathbf{C}}_4 \end{bmatrix} = \begin{bmatrix} 1^0 & 1^1 & 1^2 & 1^3 \\ 2^0 & 2^1 & 2^2 & 2^3 \\ 3^0 & 3^1 & 3^2 & 3^3 \\ 4^0 & 4^1 & 4^2 & 4^3 \end{bmatrix} \begin{bmatrix} \mathbf{A}_0^T\mathbf{B}_0 \\ \mathbf{A}_1^T\mathbf{B}_0 \\ \mathbf{A}_0^T\mathbf{B}_1 \\ \mathbf{A}_1^T\mathbf{B}_1 \end{bmatrix}.$$

The coefficient matrix in the above equation is a Vandermonde matrix, which is non-singular over reals Horn and Johnson (1991). Therefore the four components $\mathbf{A}_0^T\mathbf{B}_0$, $\mathbf{A}_1^T\mathbf{B}_0$, $\mathbf{A}_0^T\mathbf{B}_1$, $\mathbf{A}_1^T\mathbf{B}_1$ can be recovered when the master node have $\hat{\mathbf{C}}_1, \hat{\mathbf{C}}_2, \hat{\mathbf{C}}_3, \hat{\mathbf{C}}_4$.

The above coded computation strategy is resilient to 1 straggler with 5 workers. On the other hand, the uncoded strategy which is resilient to 1 straggler, each component of $\mathbf{C}$ has to run 2 copies. Therefore it needs 8 workers.

The tutorial paper Ramamoorthy et al. (2020) overviews recent developments in the field of coding for straggler-resilient distributed matrix computations.

1.2.1 Erasure coding for distributed matrix multiplication for matrices with bounded entries

A key metric of distributed matrix multiplication is the minimum number of workers that the master needs to wait for in order to compute $\mathbf{C}$; this is called the recovery threshold of the scheme. In the third part of dissertation, we present a novel coding strategy for this problem when the absolute values of the matrix entries are sufficiently small. We demonstrate a trade-off between the assumed absolute value bounds on the matrix entries and the recovery threshold. At one extreme, we are optimal with respect to the recovery threshold and on the other extreme, we match the threshold of prior work. Experimental results on cloud-based clusters validate the benefits of our method. This work has appeared in Tang et al. (2019).

1.2.2 Numerically stable coded matrix computations via circulant and rotation matrix embeddings

Ideas from coding theory have recently been used in several works for mitigating the effect of stragglers in distributed matrix computations (matrix-vector and matrix-matrix multiplication) over the reals. In particular, a polynomial code based approach spreads out the computation of $\mathbf{A}^T\mathbf{B}$ among n distributed worker nodes by means of polynomial evaluations Yu et al. (2020). This allows for an "optimal" recovery threshold whereby the intended result can be decoded as long as at least $(n-s)$ worker nodes complete their tasks; s is the number of stragglers that the scheme can handle. However, a major issue with these approaches is the high condition number of the corresponding Vandermonde-structured recovery matrices. This presents serious numerical precision issues when decoding the desired result.

It can be shown that the condition number of $n \times n$ real Vandermonde matrices grows exponentially in n Pan (2016). On the other hand, the condition numbers of Vandermonde matrices with parameters on the unit circle are much better behaved. However, using complex evaluation points in a straightforward manner causes an unacceptable multiplicative increase in the computation time at the worker nodes. In this work we leverage the properties of circulant permutation matrices and rotation matrices to obtain coded computation schemes with significantly lower worst case condition numbers; these matrices have eigenvalues that lie on the unit circle. Our technique essentially works by evaluating polynomials at matrices rather than scalars. Our analysis demonstrates that the associated recovery matrices have a condition number corresponding to Vandermonde matrices with parameters given by the eigenvalues of the corresponding circulant permutation and rotation matrices. Finally, we demonstrate an upper bound on the worst case condition number of these matrices which grows as $\approx O(n^{s+6})$. In essence, we leverage the well-behaved conditioning of complex

Vandermonde matrices with parameters on the unit circle, while still working with computation over the reals. Experimental results demonstrate that our proposed method has condition numbers that are several orders of magnitude better than prior work.

Our work in this area is currently published as a preprint Ramamoorthy and Tang (2019) and will be submitted to a journal in due course of time.

CHAPTER 2. CODED CACHING FOR NETWORKS WITH THE RESOLVABILITY PROPERTY

In this chapter, we consider the coded caching problem in a more general setting where there is a layer of relay nodes between the server and the users (see Ji et al. (2015b) for related work). Specifically, the server is connected to a set of relay nodes and the users are connected to certain subsets of the relay nodes. A class of such networks have been studied in network coding and are referred to as "combination networks" Ngai and Yeung (2004). In a combination network there are h relay nodes and $\binom{h}{r}$ users each of which is connected to a r-subset of the relay nodes. Combination networks were the first example where an unbounded gap between network coding and routing for the case of multicast was shown. While this setting is still far from an arbitrary network connecting the server and the users, it is rich enough to admit several complex strategies that may shed light on coded caching for general networks.

In this chapter we consider a class of networks that satisfy a so called resolvability property. These networks include combination networks where r divides h, but there are many other examples. We propose a coded caching scheme for these networks and demonstrate its advantages.

This chapter is organized as follows. Section 2.1 presents the problem formulation, background and our main contribution. In Section 2.2, we describe our proposed coded caching scheme, Section 2.3 presents a performance analysis and comparison and Section 2.4 concludes the paper.

2.1 Problem Formulation, Background and Main Contribution

In this work we consider a class of networks that contain an intermediate layer of nodes between the main server and the end user nodes. Combination networks are a specific type of such networks and have been studied in some depth in the literature on network coding Ngai and Yeung

(2004). However, as explained below, we actually consider a larger class of networks that encompass combination networks.

2.1.1 Problem formulation and background

The networks we consider consist of a server node denoted $\mathcal{S}$ and h relay nodes, $\Gamma_1, \Gamma_2, \dots, \Gamma_h$ such that the server is connected to each of the relay nodes by a single edge. The set of relay nodes is denoted by $\mathcal{H}$. Let $[m] = \{1, 2, \dots, m\}$. If $A \subset [h]$ we let $\Gamma_A = \cup_{i \in A} \{\Gamma_i\}$. There are K users in the system and each user is connected to a subset of $\mathcal{H}$ of size r. Let $\mathcal{V} \subset \{1, \dots, h\}$ with $|\mathcal{V}| = r$. For convenience, we will assume that the set $\mathcal{V}$ is written in ascending order even though the subset structure does not impose any ordering on the elements. Under this condition, we let $\mathcal{V}[i]$ represent the i-th element of $\mathcal{V}$. For instance, if $\mathcal{V} = \{1, 3\}$, then $\mathcal{V}[1] = 1$ and $\mathcal{V}[2] = 3$. Likewise, $\mathrm{Inv} - \mathcal{V}$ will denote the corresponding inverse map, i.e. $\mathrm{Inv} - \mathcal{V}[i] = j$ if $\mathcal{V}[j] = i$.

Each user is labeled by the subset of relay nodes it is connected to. Thus, $U_{\mathcal{V}}$ denotes the user that is connected to $\Gamma_{\mathcal{V}}$. The set of all users is denoted $\mathcal{U}$ and the set of all subsets that specify the users is denoted $\mathbf{V}$, i.e., $\mathcal{V} \in \mathbf{V}$ if $U_{\mathcal{V}}$ is a user. We consider networks where $\mathbf{V}$ satisfies the resolvability property that is defined below.

Definition 1. *Resolvability property.* The set $\mathbf{V}$ defined above is said to be resolvable if there exists a partition of $\mathbf{V}$ into subsets $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_{\tilde{K}}$ such that

- for any $i \in [\tilde{K}]$ if $\mathcal{V} \in \mathcal{P}_i$ and $\mathcal{V}' \in \mathcal{P}_i$, then $\mathcal{V} \cap \mathcal{V}' = \emptyset$, and
- for any $i \in [\tilde{K}]$, we have $\cup_{\mathcal{V}:\mathcal{V} \in \mathcal{P}_i} \mathcal{V} = [h]$.

The subsets $\mathcal{P}_i$ are referred to as the parallel classes of $\mathbf{V}$.

Each relay node Γ_i is thus connected to a set of users that is denoted by $\mathcal{N}(\Gamma_i)$. A simple counting argument shows that $|\mathcal{N}(\Gamma_i)| = Kr/h = \tilde{K}$.

Suppose that r divides h and let $\mathbf{V}$ be the set of all subsets of size r of $[h]$. In this case, the network defined above is the combination network Ji et al. (2015b) with $K = \binom{h}{r}$ users. The fact that this network satisfies the resolvability property is not obvious and follows from a result of Baranyai (1975).

Example 3. The combination network for the case of $h = 4, r = 2$ is shown in Fig. 2.1 and the corresponding parallel classes are

$$\mathcal{P}_1 = \{\{1,2\},\{3,4\}\},$$
$$\mathcal{P}_2 = \{\{1,3\},\{2,4\}\}, \text{ and}$$
$$\mathcal{P}_3 = \{\{1,4\},\{2,3\}\}.$$

On the other hand, there are other networks where $|\mathbf{V}|$ is strictly smaller than $\binom{h}{r}$.

Example 4. Let $h = 9, r = 3$ and let $\mathbf{V} = \{\{1,2,3\},\{4,5,6\},\{7,8,9\},\{1,4,7\},\{2,5,8\},\{3,6,9\}\}$. In this case, the parallel classes are

$$\mathcal{P}_1 = \{\{1,2,3\},\{4,5,6\},\{7,8,9\}\}, \text{ and}$$
$$\mathcal{P}_2 = \{\{1,4,7\},\{2,5,8\},\{3,6,9\}\}.$$

We discuss generalizations of these examples in Section III.

The server $\mathcal{S}$ contains a library of N files where each file is of size F bits (we will interchangeably refer to F as the subpacketization level). The files are represented by random variables $W_i, i = 1, \ldots, N$, where W_i is distributed uniformly over the set $[2^F]$. Each user has a cache of size MF bits. There are two distinct phases in the coded caching problem. In the placement phase, the content of the user's caches is populated. This phase should not depend on the file requests of the users. In the delivery phase, user $U_{\mathcal{V}}$ requests a file denoted $W_{d_{\mathcal{V}}}$ from the library; the set of all user requests is denoted $\mathcal{D} = \{W_{d_{\mathcal{V}}} : U_{\mathcal{V}} \text{ is a user}\}$. It can be observed that there are a total of N^K distinct request sets. The server responds by transmitting a certain number of bits that satisfies the demands of all the users. A (M, R_1, R_2) caching system also requires the specification of the following encoding and decoding functions.

- K *caching functions:* $Z_{\mathcal{V}} = \phi_{\mathcal{V}}(W_1, \ldots, W_N)$ which represents the cache content of user $U_{\mathcal{V}}$. Here, $\phi_{\mathcal{V}} : [2^{NF}] \to [2^{MF}]$.

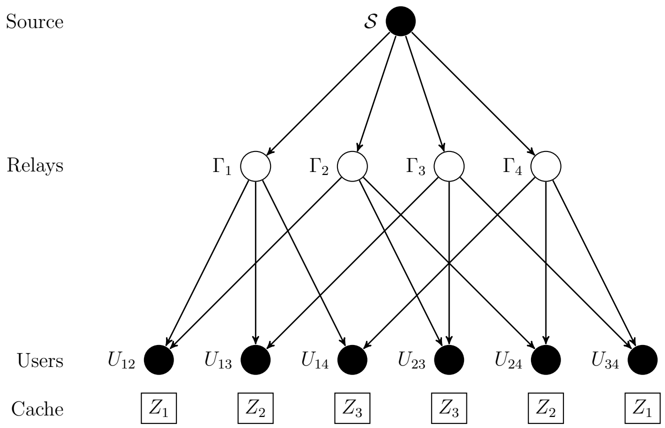

Figure 2.1 The figure shows a $\binom{4}{2}$ combination network. It also shows the cache placement when $M = 2, N = 6$. Here, $Z_1 = \cup_{n=1}^{6}\{W_{n,1}^1, W_{n,1}^2\}, Z_2 = \cup_{n=1}^{6}\{W_{n,2}^1, W_{n,2}^2\}$ and $Z_3 = \cup_{n=1}^{6}\{W_{n,3}^1, W_{n,3}^2\}$. It can be observed that each relay node sees the same caching pattern in the users that it is connected to, i.e., the users connected to each Γ_i together have Z_1, Z_2 and Z_3 represented in their caches.

- hN^K *server to relay encoding functions:* The signal $\psi_{\mathcal{S}\to\Gamma_i,\mathcal{D}}(W_1,\ldots,W_N)$ is the encoding function for the edge from $\mathcal{S}$ to relay node Γ_i. Here, $\psi_{\mathcal{S}\to\Gamma_i,\mathcal{D}} : [2^{NF}] \to [2^{R_1F}]$, so that the rate of transmission on server to relay edges is at most R_1. The signal on the edge is denoted $X_{\mathcal{S}\to\Gamma_i}$.

- $h\tilde{K}N^K$ *relay to user encoding functions:* Let $U_{\mathcal{V}} \in \mathcal{N}(\Gamma_i)$. The signal $\varphi_{\Gamma_i\to U_{\mathcal{V}},\mathcal{D}}(\psi_{\mathcal{S}\to\Gamma_i,\mathcal{D}}(W_1,\ldots,W_N))$ is the encoding function for the edge $\Gamma_i \to U_{\mathcal{V}}$. Here $\varphi_{\Gamma_i\to U_{\mathcal{V}},\mathcal{D}} : [2^{R_1F}] \to [2^{R_2F}]$, so that the rate of transmission on the relay to user edges is at most R_2. The signal on the corresponding edge is denoted $X_{\Gamma_i\to U_{\mathcal{V}}}$, which is assumed to be defined only if $U_{\mathcal{V}} \in \mathcal{N}(\Gamma_i)$.

- KN^K *decoding functions:* Every user has a decoding function for a specific request set $\mathcal{D}$, denoted by $\mu_{\mathcal{D},U_{\mathcal{V}}}(X_{\Gamma_{\mathcal{V}[1]}\to U_{\mathcal{V}}},\ldots,X_{\Gamma_{\mathcal{V}[r]}\to U_{\mathcal{V}}}, Z_{\mathcal{V}})$. Here $\mu_{\mathcal{D},U_{\mathcal{V}}} : [2^{R_2F}] \times \ldots [2^{R_2F}] \times [2^{MF}] \to [2^F]$. The decoded file is denoted by $\hat{W}_{\mathcal{D},U_{\mathcal{V}}}$.

For this coded caching system, the probability of error P_e is defined as $P_e = \max_{\mathcal{D}} \max_{\mathcal{V}} P(\hat{W}_{\mathcal{D},U_{\mathcal{V}}} \neq W_{\mathcal{D}})$.

The triplet (M, R_1, R_2) is said to be achievable if for every $\epsilon > 0$ and every large enough file size F there exists a (M, R_1, R_2) caching scheme with probability of error less than ϵ. The subpacketization level of a scheme, i.e., F is also an important metric, because it is directly connected to the implementation complexity of the scheme. For example, the original scheme of Maddah-Ali and Niesen (2014b) that operates when there is a shared link between the server and the users. It operates with a subpacketization level $F \approx \binom{K}{KM/N}$ which grows exponentially with K. Thus, the scheme of Maddah-Ali and Niesen (2014b) is applicable when the files are very large. In general, lower subpacketization levels for a given rate are preferable.

Prior work has presented two coded caching schemes for combination networks. In the routing scheme, coding is not considered. Each user simply caches a M/N fraction of each file. In the delivery phase, the total number of bits that need to be transmitted is $K(1-M/N)F$. As there are h outgoing edges from $\mathcal{S}$, we have that $R_1 = \frac{K}{h}(1 - \frac{M}{N})$. Moreover, as there are r incoming edges into each user, $R_2 = \frac{1}{r}(1 - \frac{M}{N})$. An alternate scheme, called the CM-CNC scheme was presented in Ji et al. (2015b) for combination networks. This scheme uses a decentralized cache placement phase, where each user randomly caches a M/N fraction of each file. In the delivery phase, the server executes the CM step where it encodes all requested files by a decentralized multicasting caching scheme. Following this, in the CNC step, the server divides each coded signal into r equal-size signals and encodes these r signals by a (h, r) binary MDS code. The h coded signals are transmitted over the h links from the server to relay nodes. The relay nodes forward the signals to the users. Thus, each user receives r coded signals and can recover its demand due to MDS property.

2.1.2 Main contributions

- Our schemes work for any network that satisfies the resolvability property. For a class of combination networks, we demonstrate that our achievable rates are strictly better than those proposed in prior work.

- The subpacketization level of our scheme is also significantly lower than competing methods. As discussed in Section 2.1, the subpacketization level of a given scheme directly correlates with the complexity of implementation.

2.2 Proposed Caching Scheme

Consider a network where $\mathbf{V}$ satisfies resolvability property and let the parallel classes be $\mathcal{P}_1, \cdots, \mathcal{P}_{\tilde{K}}$. It is evident that each user belongs to exactly one parallel class. Let $\Delta(\mathcal{V})$ indicate the parallel class that a user belongs to, i.e., $\Delta(\mathcal{V}) = j$ if user $U_{\mathcal{V}}$ belong to $\mathcal{P}_j$. Now, recall that $\mathcal{N}(\Gamma_i)$ is the set of users $U_{\mathcal{V}}$ such that $i \in \mathcal{V}$. By the resolvability property it has to be the case that each user in $\mathcal{N}(\Gamma_i)$ belongs to a different parallel class. In fact, it can be observed that

$$\{\Delta(\mathcal{V}) : U_{\mathcal{V}} \in \mathcal{N}(\Gamma_i)\} = [\tilde{K}], \text{ for all } i. \tag{2.1}$$

This implies that each relay node "sees" exactly the same set of parallel classes represented in the users that it is connected to. This observation inspires our placement phase in the caching scheme. We populate the user caches based on the parallel class that a given user belongs to. Loosely speaking, it turns out that we can design a symmetric uncoded placement such that the overall cache content seen by every relay node is the same.

Our proposed placement and delivery phase schemes are formally specified in Algorithm 1 and are discussed below. Assume each user has a storage capacity of $M \in \{0, \frac{N}{\tilde{K}}, \frac{2N}{\tilde{K}}, \cdots, N\}$ files, and let $t = \tilde{K}\frac{M}{N} = \frac{KrM}{hN}$. The users can be partitioned into $\tilde{K}$ groups $\mathcal{G}_i$ where $\mathcal{G}_i = \{U_{\mathcal{V}} : \mathcal{V} \in \mathcal{P}_i\}$.

In placement phase, each file W_n is split into $r\binom{\tilde{K}}{t}$ non-overlapping subfiles of equal size that are labeled as

$$W_n = (W_{n,\mathcal{T}}^l : \mathcal{T} \subset [\tilde{K}], |\mathcal{T}| = t, l \in [r]).$$

Thus, the subpacketization mechanism is such that each subfile has a superscript in $[r]$ in addition to the subset-based subscript that was introduced in the work of Maddah-Ali and Niesen (2014b).

Subfile $W_{n,\mathcal{T}}^l$ is placed in the cache of the users in $\mathcal{G}_i$ if $i \in \mathcal{T}$. Equivalently, $W_{n,\mathcal{T}}^l$ is stored in user $U_{\mathcal{V}}$ if $\Delta(\mathcal{V}) \in \mathcal{T}$. Thus, each user caches a total of $Nr\binom{\tilde{K}-1}{t-1}$ subfiles, and each subfile has size

Algorithm 1: Coded Caching in Networks Satisfying Resolvability Property

```
1. procedure: PLACEMENT PHASE ;
for $i = 1$ to $N$ do
    Partition $W_i$ into $(W^l_{n,\mathcal{T}} : \mathcal{T} \subset [\tilde{K}], |\mathcal{T}| = t, l \in [r])$
end
for $\mathcal{V} \in \mathbf{V}$ do
    $U_{\mathcal{V}}$ caches $W^l_{n,\mathcal{T}}$ if $\Delta(\mathcal{V}) \in \mathcal{T}$ for $l \in [r]$ and $n \in [N]$.
end
2.procedure: DELIVERY PHASE ;
for $i = 1$ to $h$ do
    $\Theta \leftarrow \{C : C \subset [\tilde{K}], |C| = t + 1\}$;
    Source sends $\{\oplus_{\{\mathcal{V}:\Delta(\mathcal{V})\in C\}} W^{\text{Inv}-\mathcal{V}[i]}_{d_{\mathcal{V}},C\backslash\{\Delta(\mathcal{V})\}} : i \in \mathcal{V}, C \in \Theta\}$ to $\Gamma_i$;
    for $j = 1$ to $\tilde{K}$ do
        $\Gamma_i$ forwards $\{\oplus_{\{\mathcal{V}:\Delta(\mathcal{V})\in C\}} W^{\text{Inv}-\mathcal{V}[i]}_{d_{\mathcal{V}},C\backslash\{\Delta(\mathcal{V})\}} : i \in \mathcal{V}, C \in \Theta, \Delta(\mathcal{V}) \in C\}$ to $U_{\mathcal{V}}$
    end
end
```

$\frac{F}{r\binom{\tilde{K}}{t}}$. This requires

$$Nr\binom{\tilde{K}-1}{t-1}\frac{F}{r\binom{\tilde{K}}{t}} = F\frac{Nt}{\tilde{K}} = MF$$

bits, demonstrating that our scheme uses only MF bits of cache memory at each user.

Example 5. Consider the combination network in Example 3 with $M = 2$, $N = 6$ and $K = 6$. In the placement phase, each file is partitioned into six subfiles $W^l_{n,i}$, $i = 1, 2, 3$, $l = 1, 2$. The cache placement is as follows.

$$\mathcal{G}_1 = \{U_{12}, U_{34}\} \text{ cache } W^1_{n,1}, W^2_{n,1};$$

$$\mathcal{G}_2 = \{U_{13}, U_{24}\} \text{ cache } W^1_{n,2}, W^2_{n,2}; \text{ and}$$

$$\mathcal{G}_3 = \{U_{14}, U_{23}\} \text{ cache } W^1_{n,3}, W^2_{n,3}.$$

Note that by eq. (2.1), we have that each relay node is connected to a user from each parallel class. Our placement scheme depends on the parallel class that user belongs to. Thus, it ensures that the overall distribution of the cache content seen by each relay node is the same. This can be seen in Fig. 2.1 for the example considered above. We note here that the routing scheme (*cf.* Section 2.1) is also applicable for this placement.

Now, we briefly outline the main idea of our achievable scheme. Our file parts are of the form $W^j_{n,\mathcal{T}}$, where for a given $\mathcal{T}$, $j \in [r]$. Note that each user is also connected to r different relay nodes in $\mathcal{H}$. Our proposed scheme is such that each user recovers a missing file part with a certain superscript from one of the relay nodes it is connected to. In particular, we convey enough information from the server to the relay nodes such that each relay node uses the scheme proposed in Maddah-Ali and Niesen (2014b) for one set of superscripts. Crucially, the symmetrization afforded by the placement scheme, allows each relay node to operate in this manner.

Theorem 1. Consider a network satisfying resolvability property with h relay nodes and K users such that each user is connected to r relay nodes. Suppose that the N files in the server and each user has cache of size $M \in \{0, \frac{Nh}{Kr}, \frac{2Nh}{Kr}, \cdots, N\}$. Then, the following rate pair (R_1, R_2) is achievable.

$$R_1 = \min\left\{\frac{K(1-\frac{M}{N})}{h(1+\frac{KrM}{hN})}, \frac{N}{r}(1-\frac{M}{N})\right\}, \tag{2.2}$$

$$R_2 = \frac{1-\frac{M}{N}}{r}.$$

For general $0 \le M \le N$, the lower convex envelope of these points is achievable.

Proof. Let the set of user requests be denoted $\mathcal{D} = \{W_{d_\mathcal{V}} : U_\mathcal{V} \text{ is a user}\}$. For each relay node Γ_i, we focus on the users connected to it $\mathcal{N}(\Gamma_i)$ and a subset $C \subset \{\Delta(\mathcal{V}) : U_\mathcal{V} \in \mathcal{N}(\Gamma_i)\}$ where $|C| = t+1$. For each subset C, the server transmits

$$\oplus_{\{\mathcal{V}:\Delta(\mathcal{V})\in C\}} W^{\text{Inv}-\mathcal{V}[i]}_{d_\mathcal{V}, C\setminus\{\Delta(\mathcal{V})\}} \tag{2.3}$$

to the relay node Γ_i ($\oplus$ denotes bitwise XOR) and Γ_i forwards it into users $U_\mathcal{V}$ where $\Delta(\mathcal{V}) \in C$.

We now argue that each user can recover its requested file. Evidently, a user $U_\mathcal{V}$ is missing subfiles of the form $W^j_{d_\mathcal{V},\mathcal{T}}$ where $\Delta(\mathcal{V}) \notin \mathcal{T}$. If $U_\mathcal{V}$ is connected to Γ_i, it can recover the following set of subfiles using the transmissions from Γ_i.

$$\{W^{\text{Inv}-\mathcal{V}[i]}_{d_\mathcal{V},\mathcal{T}} : \mathcal{T} \subset [\tilde{K}] \setminus \{\Delta(\mathcal{V})\}, |\mathcal{T}| = t\}.$$

This is because the transmission in eq. (2.3) is such that $U_{\mathcal{V}}$ caches all subfiles that are involved in the XOR except the one that is interested in. This implies it can decode its missing subfile. In addition, $U_{\mathcal{V}}$ is also connected to r relay nodes so that $\cup_{i\in\mathcal{V}}\{\mathrm{Inv}-\mathcal{V}[i]\} = [r]$, i.e., it can recover all its missing subfiles.

Next, we determine R_1 and R_2. Each of coded subfiles results in $\frac{F}{r\binom{\tilde{K}}{t}}$ bits being sent over the link from source $\mathcal{S}$ to Γ_i. Since the number of subsets C is $\binom{\tilde{K}}{t+1}$, the total number of bits sent from $\mathcal{S}$ to Γ_i is $\binom{\tilde{K}}{t+1}\frac{F}{r\binom{\tilde{K}}{t}} = F\frac{\tilde{K}(1-\frac{M}{N})}{r(1+\frac{\tilde{K}M}{N})}$, and hence $R_1 = \frac{\tilde{K}(1-\frac{M}{N})}{r(1+\frac{\tilde{K}M}{N})}$. Next, note that each coded subfile is forwarded to $|C| = t+1$ users. Thus, each user receives $\frac{|C|\binom{\tilde{K}}{t+1}}{\tilde{K}}$ coded subfiles so that the total number of bits sent from a relay node to user is $\frac{|C|\binom{\tilde{K}}{t+1}}{\tilde{K}} \times \frac{F}{r\binom{\tilde{K}}{t}} = \frac{|C|F(\tilde{K}-t)}{r\tilde{K}(t+1)} = F\frac{1-\frac{M}{N}}{r}$. Hence $R_2 = \frac{1-\frac{M}{N}}{r}$.

Thus, the triplet $(M, R_1, R_2) = (M, \frac{\tilde{K}(1-\frac{M}{N})}{r(1+\frac{\tilde{K}M}{N})}, \frac{1-\frac{M}{N}}{r})$ is achievable for $M \in \{0, \frac{Nh}{Kr}, \frac{2Nh}{Kr}, \cdots, N\}$. Points for general values of M can be obtained by memory sharing between triplets of this form.

If $\tilde{K} > N$, it is clear that some users connecting to a given relay node Γ_i request the same file. In this case, the routing scheme can attain $R_1 = \frac{N}{r}(1-\frac{M}{N})$, which is better than the proposed scheme if $M \leq 1 - \frac{N}{\tilde{K}}$. This explains the second term within the minimum in the RHS of eq. (2.2). □

We illustrate our achievable scheme by considering the setup in Example 5.

Example 6. Assume that user $U_{\mathcal{V}}, \mathcal{V} \subset \{1,\ldots,4\}, |\mathcal{V}| = 2$, requires file $W_{d_{\mathcal{V}}}$. The users connected to Γ_1 correspond to subsets $\{1,2\}, \{1,3\}$ and $\{1,4\}$ so that $\mathrm{Inv}-\mathcal{V}[1] = 1$ for all of them. Thus, the users recover missing subfiles with superscript of 1 from Γ_1. In particular, the transmissions are as follows.

$$S \to \Gamma_1 : W^1_{d_{12},2} \oplus W^1_{d_{13},1}, W^1_{d_{12},3} \oplus W^1_{d_{14},1}, W^1_{d_{13},3} \oplus W^1_{d_{14},2},$$

$$\Gamma_1 \to U_{12} : W^1_{d_{12},2} \oplus W^1_{d_{13},1}, W^1_{d_{12},3} \oplus W^1_{d_{14},1},$$

$$\Gamma_1 \to U_{13} : W^1_{d_{12},2} \oplus W^1_{d_{13},1}, W^1_{d_{13},3} \oplus W^1_{d_{14},2}, \text{ and}$$

$$\Gamma_1 \to U_{14} : W^1_{d_{12},3} \oplus W^1_{d_{14},1}, W^1_{d_{13},3} \oplus W^1_{d_{14},2}.$$

The users connected to Γ_2 correspond to subsets $\{1,2\},\{2,3\}$ and $\{2,4\}$ in which case Inv $-$ $\{1,2\}[2] = 2$ while Inv $-$ $\{2,3\}[2] = 1$, Inv $-$ $\{2,4\}[2] = 1$. Thus, user U_{12} recovers missing subfiles with superscript 2 from Γ_2 while users U_{23} and U_{24} recover missing subfiles with superscript 1. The specific transmissions are given below.

$$\begin{aligned}
&S \to \Gamma_2 : W^1_{d_{23},1} \oplus W^2_{d_{12},3}, W^2_{d_{12},2} \oplus W^1_{d_{24},1}, W^1_{d_{23},2} \oplus W^1_{d_{24},3},\\
&\Gamma_2 \to U_{12} : W^1_{d_{23},1} \oplus W^2_{d_{12},3}, W^2_{d_{12},2} \oplus W^1_{d_{24},1},\\
&\Gamma_2 \to U_{23} : W^1_{d_{23},1} \oplus W^2_{d_{12},3}, W^1_{d_{23},2} \oplus W^1_{d_{24},3}, \text{ and}\\
&\Gamma_2 \to U_{24} : W^2_{d_{12},2} \oplus W^1_{d_{24},1}, W^1_{d_{23},2} \oplus W^1_{d_{24},3}.
\end{aligned}$$

In a similar manner, the other transmissions can be determined and it can be verified that the demands of the users are satisfied and $R_1 = \frac{1}{2}$, $R_2 = \frac{1}{3}$.

It is important to note that the resolvability property is key to our proposed scheme. For example, if r does not divide h, the combination network does not have the resolvability property. In this case, it can be shown that a symmetric uncoded placement is impossible. We demonstrate this by means of the example below.

Example 7. Consider the combination network with $h = 3, r = 2$, and $\mathbf{V} = \{\{1,2\},\{1,3\},\{2,3\}\}$. Here 2 does not divide 3 and it is easy to check that it does not satisfy the resolvability property. Next, we argue that a symmetric uncoded placement is impossible by contradiction. Assume there exists a symmetric uncoded placement and suppose U_{12} caches Z_1, U_{13} caches Z_2. By the hypothesis, Γ_1 and Γ_2 have to see the same cache content. Since $\mathcal{N}(\Gamma_1) = \{U_{12}, U_{13}\}$ and $\mathcal{N}(\Gamma_2) = \{U_{12}, U_{23}\}$, U_{23} has to cache Z_2. As a result, since $\mathcal{N}(\Gamma_3) = \{U_{13}, U_{23}\}$, Γ_3 sees Z_2 and Z_2, which are different from the cache content seen by Γ_1 and Γ_2. This is a contradiction.

We emphasize that a large class of networks satisfy the resolvability property. For instance, if r divides h, Baranyai (1975) shows that the set of all $\binom{h}{r}$ r-subsets of an h-set can be partitioned into disjoint parallel classes $\mathcal{P}_i$, $i = 1, 2, \cdots, \binom{h-1}{r-1}$. More generally, one can consider resolvable designs Stinson (2003) which are set systems that satisfy the resolvability property. Such designs include

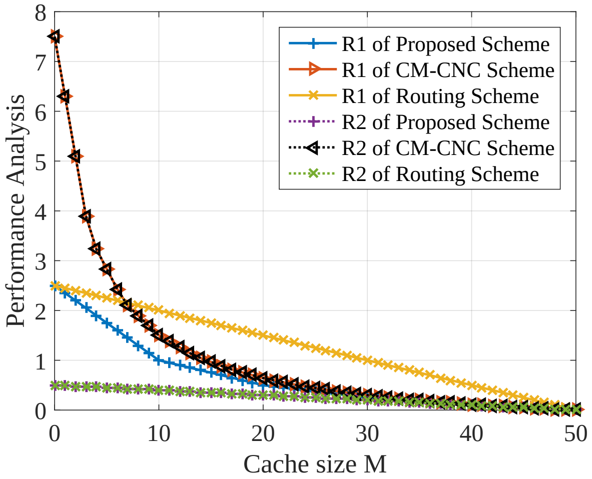

Figure 2.2 Performance comparison of the different schemes for a $\binom{6}{2}$ combination network with $K = 15$, $\tilde{K} = 5$ and $N = 50$.

affine planes which correspond to networks where for prime q, we have $h = q^2$ and the $r = q$; the set $\mathbf{V}$ is given by the specification of the affine plane. Furthermore, one can obtain resolvable designs from affine geometry over $\mathbb{F}_q$ that will correspond to networks with $h = q^m$ and $r = q^d$.

2.3 Performance Analysis

We now compare the performance of our proposed scheme with the CM-CNC scheme Ji et al. (2015b) and the routing scheme. For a given value of M we compare the achievable R_1, R_2 pairs of the different schemes. Furthermore, we also compare the required subpacketization levels of the different schemes, as it directly impacts the complexity of implementation of a given scheme. Table 2.1 summarizes the comparison. We note here that the rate of the CM-CNC scheme is derived in Ji et al. (2015b) for a decentralized placement. The rate in Table 2.1 corresponds to a derivation of the corresponding rate for a centralized placement; it is lower than the one for the decentralized placement.

Table 2.1 Comparison of three schemes

	Routing	CM-CNC	New Scheme
F	$r\binom{\tilde{K}}{\frac{\tilde{K}M}{N}}$	$r\binom{K}{\frac{KM}{N}}$	$r\binom{\tilde{K}}{\frac{\tilde{K}M}{N}}$
R_1	$\frac{K}{h}(1-\frac{M}{N})$	$\frac{K(1-\frac{M}{N})}{r(1+\frac{KM}{N})}$	$\frac{\tilde{K}(1-\frac{M}{N})}{r(1+\frac{\tilde{K}M}{N})}$
R_2	$\frac{1}{r}(1-\frac{M}{N})$	$\frac{K(1-\frac{M}{N})}{r(1+\frac{KM}{N})}$	$\frac{1}{r}(1-\frac{M}{N})$

The following conclusions can be drawn. Let (R_1^*, R_2^*) and F^* denote the rates and subpacketization level of our proposed scheme.

$$\begin{aligned}\frac{R_1^*}{R_1^{CM-CNC}} &= \frac{\frac{1}{K}+\frac{M}{N}}{\frac{1}{\tilde{K}}+\frac{M}{N}} < 1, \text{ and}\\ \frac{R_2^*}{R_2^{CM-CNC}} &= \frac{1}{K}+\frac{M}{N}\\ &\leq \frac{N-\frac{N}{\tilde{K}}}{N}+\frac{1}{K} < 1.\end{aligned}$$

This implies that our scheme is better in both rate metrics. Next,

$$\frac{F^*}{F^{CM-CNC}} \approx \exp\left\{(K(1-\frac{r}{h})H_e(\frac{M}{N})\right\}$$

where $H_e(\cdot)$ represents the binary entropy function in nats. Thus, the subpacketization level of our scheme is exponentially smaller than the scheme of Ji et al. (2015b).

For a $\binom{6}{2}$ combination network with parameters $K = 15$, $\tilde{K} = 5$, $N = 50$, we plot the performance of the different schemes in Fig. 2.2. Fig. 2.2 compares R_1 and R_2 of three schemes. It can be observed that for R_1, the proposed scheme is best for all cache size M. At the same time, we can see that R_2 of routing scheme and the proposed scheme are identical but significantly better than that of CM-CNC scheme.

2.4 Conclusions

In this work, we proposed a coding caching scheme for networks that satisfy the resolvability property. This family of networks includes a class of combination networks as a special case. The rate required by our scheme for transmission over the server-to-relay edges and over the relay-to-user edges is strictly lesser than that proposed in prior work. In addition, the subpacketization level of our scheme is also significantly lower than prior work. The generalization to networks that do not satisfy the resolvability property and to networks with arbitrary topologies is an interesting direction for future work.

CHAPTER 3. CODED CACHING SCHEMES WITH REDUCED SUBPACKETIZATION FROM LINEAR BLOCK CODES

In this chapter, we examine an important aspect of the coded caching problem that is closely tied to its adoption in practice. It is important to note that the huge gains of coded caching require each file to be partitioned into $F_s \approx \binom{K}{\frac{KM}{N}}$ non-overlapping subfiles of equal size; F_s is referred to as the *subpacketization level*. It can be observed that for a fixed cache size $\frac{M}{N}$, F_s grows exponentially with K. This can be problematic in practical implementations. For instance, suppose that $K = 64$, with $\frac{M}{N} = 0.25$ so that $F_s = \binom{64}{16} \approx 4.8 \times 10^{14}$ with a rate $R \approx 2.82$. In this case, it is evident that at the bare minimum, the size of each file has to be at least 480 terabits for leveraging the gains in Maddah-Ali and Niesen (2014b). It is even worse in practice. The atomic unit of storage on present day hard drives is a sector of size 512 bytes and the trend in the disk drive industry is to move this to 4096 bytes Fitzpatrick (2011). As a result, the minimum size of each file needs to be much higher than 480 terabits. Therefore, the scheme in Maddah-Ali and Niesen (2014b) is not practical even for moderate values of K. Furthermore, even for smaller values of K, schemes with low subpacketization levels are desirable. This is because any practical scheme will require each of the subfiles to have some header information that allows for decoding at the end users. When there are a large number of subfiles, the header overhead may be non-negligible. For these same parameters ($K = 64, M/N = 0.25$) our proposed approach in this work allows us obtain, e.g., the following operating points: (i) $F_s \approx 1.07 \times 10^9$ and $R = 3$, (ii) $F_s \approx 1.6 \times 10^4$ and $R = 6$, (iii) $F_s = 64$ and $R = 12$. For the first point, it is evident that the subpacketization level drops by over five orders of magnitude with only a very small increase in the rate. Point (ii) and (iii) show proposed scheme allows us to operate at various points on the trade-off between subpacketization level and rate.

The issue of subpacketization was first considered in the work of Shanmugam et al. (2014, 2016) in the decentralized coded caching setting. In the centralized case it was considered in the work of Yan et al. (2017a). They proposed a low subpacketization scheme based on placement delivery arrays. Reference Shangguan et al. (2018) viewed the problem from a hypergraph perspective and presented several classes of coded caching schemes. The work of Shanmugam et al. (2017) has recently shown that there exist coded caching schemes where the subpacketization level grows linearly with the number of users K; however, this result only applies when the number of users is very large. We elaborate on related work in Section 3.1.1.

In this work, we propose low subpacketization level schemes for coded caching. Our proposed schemes leverage the properties of combinatorial structures known as resolvable designs and their natural relationship with linear block codes. Our schemes are applicable for a wide variety of parameter ranges and allow the system designer to tune the subpacketization level and the gain of the system with respect to an uncoded system. We note here that designs have also been used to obtain results in distributed data storage Olmez and Ramamoorthy (2016) and network coding based function computation in recent work Tripathy and Ramamoorthy (2015, 2017).

This chapter is organized as follows. Section 3.1 discusses the background and related work and summarizes the main contributions of our work. Section 3.2 outlines our proposed scheme. It includes all the constructions and the essential proofs. A central object of study in our work are matrices that satisfy a property that we call the consecutive column property (CCP). Section 3.3 overviews several constructions of matrices that satisfy this property. Several of the longer and more involved proofs of statements in Sections 3.2 and 3.3 appear in the Appendix. In Section 3.4 we perform an in-depth comparison our work with existing constructions in the literature. We conclude the paper with a discussion of opportunities for future work in Section 3.5.

3.1 Background, Related Work and Summary of Contributions

We consider a scenario where the server has N files each of which consist of F_s subfiles. There are K users each equipped with a cache of size MF_s subfiles. The coded caching scheme is specified by means of the placement scheme and an appropriate delivery scheme for each possible demand pattern. In this work, we use combinatorial designs Stinson (2003) to specify the placement scheme in the coded caching system.

Definition 2. A design is a pair $(X, \mathcal{A})$ such that

1. X is a set of elements called points, and
2. $\mathcal{A}$ is a collection of nonempty subsets of X called blocks, where each block contains the same number of points.

A design is in one-to-one correspondence with an incidence matrix $\mathcal{N}$ which is defined as follows.

Definition 3. The incidence matrix $\mathcal{N}$ of a design $(X, \mathcal{A})$ is a binary matrix of dimension $|X| \times |\mathcal{A}|$, where the rows and columns correspond to the points and blocks respectively. Let $i \in X$ and $j \in \mathcal{A}$. Then,

$$\mathcal{N}(i,j) = \begin{cases} 1 & \text{if } i \in j, \\ 0 & \text{otherwise.} \end{cases}$$

It can be observed that the transpose of an incidence matrix also specifies a design. We will refer to this as the transposed design. In this work, we will utilize resolvable designs which are a special class of designs.

Definition 4. A parallel class $\mathcal{P}$ in a design $(X, \mathcal{A})$ is a subset of disjoint blocks from $\mathcal{A}$ whose union is X. A partition of $\mathcal{A}$ into several parallel classes is called a resolution, and $(X, \mathcal{A})$ is said to be a resolvable design if $\mathcal{A}$ has at least one resolution.

For resolvable designs, it follows that each point also appears in the same number of blocks.

Example 8. Consider a block design specified as follows.

$$X = \{1, 2, 3, 4\}, \text{ and}$$

$$\mathcal{A} = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}.$$

Its incidence matrix is given below.

$$\mathcal{N} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

It can be observed that this design is resolvable with the following parallel classes.

$$\mathcal{P}_1 = \{\{1,2\},\{3,4\}\},$$
$$\mathcal{P}_2 = \{\{1,3\},\{2,4\}\}, \text{ and}$$
$$\mathcal{P}_3 = \{\{1,4\},\{2,3\}\}.$$

In the sequel we let $[n]$ denote the set $\{1,\dots,n\}$. We emphasize here that the original scheme of Maddah-Ali and Niesen (2014b) can be viewed as an instance of the trivial design. For example, consider the setting when $t = KM/N$ is an integer. Let $X = [K]$ and $\mathcal{A} = \{B : B \subset [K], |B| = t\}$. In the scheme of Maddah-Ali and Niesen (2014b), the users are associated with X and the subfiles with $\mathcal{A}$. User $i \in [K]$ caches subfile $W_{n,B}, n \in [N]$ for $B \in \mathcal{A}$ if $i \in B$. The main message of our work is that carefully constructed resolvable designs can be used to obtain coded caching schemes with low subpacketization levels, while retaining much of the rate gains of coded caching. The basic idea is to associate the users with the blocks and the subfiles with the points of the design. The roles of the users and subfiles can also be interchanged by simply working with the transposed design.

Example 9. Consider the resolvable design from Example 8. The blocks in $\mathcal{A}$ correspond to six users $U_{12}, U_{34}, U_{13}, U_{24}, U_{14}, U_{23}$. Each file is partitioned into $F_s = 4$ subfiles $W_{n,1}, W_{n,2}, W_{n,3}, W_{n,4}$ which correspond to the four points in X. The cache in user U_B, denoted Z_B is specified as $Z_{ij} = (W_{n,i}, W_{n,j})_{n=1}^{N}$. For example, $Z_{12} = (W_{n,1}, W_{n,2})_{n=1}^{N}$.

We note here that the caching scheme is symmetric with respect to the files in the server. Furthermore, each user caches half of each file so that $M/N = 1/2$. Suppose that in the delivery phase user U_B requests file W_{d_B} where $d_B \in [N]$. These demands can be satisfied as follows. We pick three blocks, one each from parallel classes $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$ and generate the signals transmitted in the delivery phase as follows.

$$\begin{aligned}
&W_{d_{12},3} \oplus W_{d_{13},2} \oplus W_{d_{23},1}, \qquad (3.1)\\
&W_{d_{12},4} \oplus W_{d_{24},1} \oplus W_{d_{14},2},\\
&W_{d_{34},1} \oplus W_{d_{13},4} \oplus W_{d_{14},3}, \text{ and}\\
&W_{d_{34},2} \oplus W_{d_{24},3} \oplus W_{d_{23},4}.
\end{aligned}$$

The three terms in the in eq. (3.1) above correspond to blocks from different parallel classes $\{1,2\} \in \mathcal{P}_1, \{1,3\} \in \mathcal{P}_2, \{2,3\} \in \mathcal{P}_3$. This equation has the *all-but-one* structure that was also exploited in Maddah-Ali and Niesen (2014b), i.e., eq. (3.1) is such that each user caches all but one of the subfiles participating in the equation. Specifically, user U_{12} contains $W_{n,1}$ and $W_{n,2}$ for all $n \in [N]$. Thus, it can decode subfile $W_{d_{12},3}$ that it needs. A similar argument applies to users U_{13} and U_{23}. It can be verified that the other three equations also have this property. Thus, at the end of the delivery phase, each user obtains its missing subfiles.

This scheme corresponds to a subpacketization level of 4 and a rate of 1. In contrast, the scheme of Maddah-Ali and Niesen (2014b) would require a subpacketization level of $\binom{6}{3} = 20$ with a rate of 0.75. Thus, it is evident that we gain significantly in terms of the subpacketization while sacrificing some rate gains.

As shown in Example 9, we can obtain a scheme by associating the users with the blocks and the subfiles with the points. In this work, we demonstrate that this basic idea can be significantly generalized and several schemes with low subpacketization levels that continue to leverage much of the rate benefits of coded caching can be obtained.

3.1.1 Discussion of Related Work

Coded caching has been the subject of much investigation in recent work as discussed briefly earlier on. We now overview existing literature on the topic of low subpacketization schemes for coded caching. In the original paper Maddah-Ali and Niesen (2014b), for given problem parameters

K (number of users) and M/N (cache fraction), the authors showed that when $N \geq K$, the rate equals

$$R = \frac{K(1 - M/N)}{1 + KM/N}$$

when M is an integer multiple of N/K. Other points are obtained via memory sharing. Thus, in the regime when KM/N is large, the coded caching rate is approximately $N/M - 1$, which is independent of K. Crucially, though this requires the subpacketization level $F_s \approx \binom{K}{KM/N}$. It can be observed that for a fixed M/N, F_s grows exponentially with K. This is one of main drawbacks of the original scheme, deploying this solution in practice may be difficult.

The subpacketization issue was first discussed in the work of Shanmugam et al. (2014, 2016) in the context of decentralized caching. Specifically, Shanmugam et al. (2016) showed that in the decentralized setting for any subpacketization level F_s such that $F_s \leq \exp(KM/N)$ the rate would scale linearly in K, i.e., $R \geq cK$. Thus, much of the rate benefits of coded caching would be lost if F_s did not scale exponentially in K. Following this work, the authors in Yan et al. (2017a) introduced a technique for designing low subpacketization schemes in the centralized setting which they called placement delivery arrays. In Yan et al. (2017a), they considered the setting when $M/N = 1/q$ or $M/N = 1 - 1/q$ and demonstrated a scheme where the subpacketization level was exponentially smaller than the original scheme, while the rate was marginally higher. This scheme can be viewed as a special case of our work. We discuss these aspects in more detail in Section 3.4. In Shangguan et al. (2018), the design of coded caching schemes was achieved through the design of hypergraphs with appropriate properties. In particular, for specific problem parameters, they were able to establish the existence of schemes where the subpacketization scaled as $\exp(c\sqrt{K})$. ReferenceYan et al. (2017b) presented results in this setting by considering strong edge coloring of bipartite graphs.

Very recently, Shanmugam et al. (2017) showed the existence of coded caching schemes where the subpacketization grows linearly with the number of users, but the coded caching rate grows as $O(K^{\delta})$ where $0 < \delta < 1$. Thus, while the rate is not a constant, it does not grow linearly with K either. Both Shangguan et al. (2018) and Shanmugam et al. (2017) are interesting results that demonstrate the existence of regimes where the subpacketization scales in a manageable manner. Nevertheless, it is to be noted that these results come with several caveats. For example, the result of Shanmugam et al. (2017) is only valid in the regime when K is very large and is unlikely to be of use for practical values of K. The result of Shangguan et al. (2018) has significant restrictions on the number of users, e.g., in their paper, K needs to be of the form $\binom{n}{a}$ and $q^t\binom{n}{a}$.

3.1.2 Summary of Contributions

In this work, the subpacketization levels we obtain are typically exponentially smaller than the original scheme. However, they still continue to scale exponentially in K, albeit with much smaller exponents. However, our construction has the advantage of being applicable for a large range of problem parameters. Our specific contributions include the following.

- We uncover a simple and natural relationship between a (n, k) linear block code and a coded caching scheme. We first show that any linear block code over $GF(q)$ and in some cases $\mathbb{Z} \mod q$ (where q is not a prime or a prime power) generates a resolvable design. This design in turn specifies a coded caching scheme with $K = nq$ users where the cache fraction $M/N = 1/q$. A complementary cache fraction point where $M/N = 1 - \alpha/nq$ where α is some integer between 1 and $k + 1$ can also be obtained. Intermediate points can be obtained by memory sharing between these points.

- We consider a class of (n, k) linear block codes whose generator matrices satisfy a specific rank property. In particular, we require collections of consecutive columns to have certain rank properties. For such codes, we are able to identify an efficient delivery phase and determine the precise coded caching rate. We demonstrate that the subpacketization level is at most $q^k(k + 1)$ whereas the coded caching gain scales as $k + 1$ with respect to an uncoded caching scheme. Thus, different choices of k allow the system designer significant flexibility to choose the appropriate operating point.

- We discuss several constructions of generator matrices that satisfy the required rank property. We characterize the ranges of alphabet sizes (q) over which these matrices can be constructed.

If one has a given subpacketization budget in a specific setting, we are able to find a set of schemes that fit the budget while leveraging the rate gains of coded caching.

3.2 Proposed low subpacketization level scheme

All our constructions of low subpacketization schemes will stem from resolvable designs (*cf.* Definition 4). Our overall approach is to first show that any (n, k) linear block code over $GF(q)$ can be used to obtain a resolvable block design. The placement scheme obtained from this resolvable design is such that $M/N = 1/q$. Under certain (mild) conditions on the generator matrix we show that a delivery phase scheme can be designed that allows for a significant rate gain over the uncoded scheme while having a subpacketization level that is significantly lower than Maddah-Ali and Niesen (2014b). Furthermore, our scheme can be transformed into another scheme that operates at the point $M/N = 1 - \frac{k+1}{nq}$. Thus, intermediate values of M/N can be obtained via memory sharing. We also discuss situations under which we can operate over modular arithmetic $\mathbb{Z}_q = \mathbb{Z} \mod q$ where q is not necessarily a prime or a prime power; this allows us to obtain a larger range of parameters.

3.2.1 Resolvable Design Construction

Consider a (n, k) linear block code over $GF(q)$. To avoid trivialities we assume that its generator matrix does not have an all-zeros column. We collect its q^k codewords and construct a matrix $\mathbf{T}$ of size $n \times q^k$ as follows.

$$\mathbf{T} = [\mathbf{c}_0^T, \mathbf{c}_1^T, \cdots, \mathbf{c}_{q^k-1}^T], \tag{3.2}$$

where the $1 \times n$ vector $\mathbf{c}_\ell$ represents the ℓ-th codeword of the code. Let $X = \{0, 1, \cdots, q^k - 1\}$ be the point set and $\mathcal{A}$ be the collection of all subsets $B_{i,l}$ for $0 \leq i \leq n-1$ and $0 \leq l \leq q-1$, where

$$B_{i,l} = \{j : \mathbf{T}_{i,j} = l\}.$$

Using this construction, we can obtain the following result.

Lemma 1. The construction procedure above results in a design $(X, \mathcal{A})$ where $X = \{0, 1, \cdots, q^k - 1\}$ and $|B_{i,l}| = q^{k-1}$ for all $0 \leq i \leq n-1$ and $0 \leq l \leq q-1$. Furthermore, the design is resolvable with parallel classes given by $\mathcal{P}_i = \{B_{i,l} : 0 \leq l \leq q-1\}$, for $0 \leq i \leq n-1$.

Proof. Let $\mathbf{G} = [g_{ab}]$, for $0 \leq a \leq k-1$, $0 \leq b \leq n-1$, $g_{ab} \in GF(q)$. Note that for $\Delta = [\Delta_0 \ \Delta_1 \ \ldots \ \Delta_{n-1}] = \mathbf{uG}$, we have

$$\Delta_b = \sum_{a=0}^{k-1} \mathbf{u}_a g_{ab},$$

where $\mathbf{u} = [\mathbf{u}_0, \cdots, \mathbf{u}_{k-1}]$. Let a^* be such that $g_{a^*b} \neq 0$. Consider the equation

$$\sum_{a \neq a^*} \mathbf{u}_a g_{ab} = \Delta_b - \mathbf{u}_{a^*} g_{a^*b},$$

where Δ_b is fixed. For arbitrary values of $\mathbf{u}_a$, $a \neq a^*$, this equation has a unique solution for $\mathbf{u}_{a^*}$, which implies that for any Δ_b, $|B_{b,\Delta_b}| = q^{k-1}$ and that $\mathcal{P}_b$ forms a parallel class. □

Remark 1. A $k \times n$ generator matrix over $GF(q)$ where q is a prime power can also be considered as a matrix over an extension field $GF(q^m)$ where m is an integer. Thus, one can obtain a resolvable design in this case as well; the corresponding parameters can be calculated in an easy manner.

Remark 2. We can also consider linear block codes over $\mathbb{Z} \mod q$ where q is not necessarily a prime or a prime power. In this case the conditions under which a resolvable design can be obtained by forming the matrix $\mathbf{T}$ are a little more involved. We discuss this in Lemma 5 in the Appendix.

Example 10. Consider a $(4, 2)$ linear block code over $GF(3)$ with generator matrix

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \end{bmatrix}.$$

Collecting the nine codewords, $\mathbf{T}$ is constructed as follows.

$$\mathbf{T} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\ 0 & 1 & 2 & 1 & 2 & 0 & 2 & 0 & 1 \\ 0 & 2 & 1 & 1 & 0 & 2 & 2 & 1 & 0 \end{bmatrix}.$$

Using $\mathbf{T}$, we generate the resolvable block design $(X, \mathcal{A})$ where the point set is $X = \{0,1,2,3,4,5,6,7,8\}$. For instance, block $B_{0,0}$ is obtained by identifying the column indexes of zeros in the first row of $\mathbf{T}$, i.e., $B_{0,0} = \{0,1,2\}$. Following this, we obtain

$$\begin{aligned}\mathcal{A} =& \{\{0,1,2\},\{3,4,5\},\{6,7,8\},\{0,3,6\},\{1,4,7\},\{2,5,8\},\\ & \{0,5,7\},\{1,3,8\},\{2,4,6\},\{0,4,8\},\{2,3,7\},\{1,5,6\}\}.\end{aligned}$$

It can be observed that $\mathcal{A}$ has a resolution (*cf.* Definition 4) with the following parallel classes.

$$\begin{aligned}\mathcal{P}_0 &= \{\{0,1,2\},\{3,4,5\},\{6,7,8\}\},\\ \mathcal{P}_1 &= \{\{0,3,6\},\{1,4,7\},\{2,5,8\}\},\\ \mathcal{P}_2 &= \{\{0,5,7\},\{1,3,8\},\{2,4,6\}\}, \text{ and}\\ \mathcal{P}_3 &= \{\{0,4,8\},\{2,3,7\},\{1,5,6\}\}.\end{aligned}$$

3.2.2 A special class of linear block codes

We now introduce a special class of linear block codes whose generator matrices satisfy specific rank properties. It turns out that resolvable designs obtained from these codes are especially suited for usage in coded caching.

Consider the generator matrix $\mathbf{G}$ of a (n, k) linear block code over $GF(q)$. The i-th column of $\mathbf{G}$ is denoted by $\mathbf{g}_i$. Let z be the least positive integer such that $k+1$ divides nz (denoted by $k+1 \mid nz$). We let $(t)_n$ denote t mod n.

In our construction we will need to consider various collections of $k+1$ consecutive columns of $\mathbf{G}$ (wraparounds over the boundaries are allowed). For this purpose, let $\mathcal{T}_a = \{a(k+1), \cdots, a(k+1)+k\}$ (a is a non-negative integer) and $\mathcal{S}_a = \{(t)_n \mid t \in \mathcal{T}_a\}$. Let $\mathbf{G}_{\mathcal{S}_a}$ be the $k \times (k+1)$ submatrix of $\mathbf{G}$ specified by the columns in $\mathcal{S}_a$, i.e., $\mathbf{g}_\ell$ is a column in $\mathbf{G}_{\mathcal{S}_a}$ if $\ell \in \mathcal{S}_a$. Next, we define the $(k, k+1)$-consecutive column property that is central to the rest of the discussion.

Definition 5. $(k, k+1)$-*consecutive column property.* Consider the submatrices of $\mathbf{G}$ specified by $\mathbf{G}_{\mathcal{S}_a}$ for $0 \le a \le \frac{zn}{k+1} - 1$. We say that $\mathbf{G}$ satisfies the $(k, k+1)$-consecutive column property if all $k \times k$ submatrices of each $\mathbf{G}_{\mathcal{S}_a}$ are full rank.

Henceforth, we abbreviate the $(k, k+1)$-consecutive column property as $(k, k+1)$-CCP.

Example 11. In Example 10 we have $k = 2, n = 4$ and hence $z = 3$. Thus, $\mathcal{S}_0 = \{0, 1, 2\}, \mathcal{S}_1 = \{3, 0, 1\}, \mathcal{S}_2 = \{2, 3, 0\}$ and $\mathcal{S}_3 = \{1, 2, 3\}$. The corresponding generator matrix $\mathbf{G}$ satisfies the $(k, k+1)$ CCP as any two columns of the each of submatrices $\mathbf{G}_{\mathcal{S}_i}, i = 0, \ldots, 3$ are linearly independent over $GF(3)$.

We note here that one can also define different levels of the consecutive column property. Let $\mathcal{T}_a^\alpha = \{a\alpha, \cdots, a\alpha + \alpha - 1\}$ and $\mathcal{S}_a^\alpha = \{(t)_n \mid t \in \mathcal{T}_a^\alpha\}$.

Definition 6. (k, α)-*consecutive column property* Consider the submatrices of $\mathbf{G}$ specified by $\mathbf{G}_{\mathcal{S}_a^\alpha}$ for $0 \leq a \leq \frac{zn}{\alpha} - 1$. We say that $\mathbf{G}$ satisfies the (k, α)-consecutive column property, where $\alpha \leq k$ if each $\mathbf{G}_{\mathcal{S}_a^\alpha}$ has full rank. In other words, the α columns in each $\mathbf{G}_{\mathcal{S}_a^\alpha}$ are linearly independent.

As pointed out in the sequel, codes that satisfy the (k, α)-CCP, where $\alpha \leq k$ will result in caching systems that have a multiplicative rate gain of α over an uncoded system. Likewise, codes that satisfy the $(k, k+1)$-CCP will have a gain of $k+1$ over an uncoded system. In the remainder of the paper, we will use the term CCP to refer to the $(k, k+1)$-CCP if the value of k is clear from the context.

3.2.3 Usage in a coded caching scenario

A resolvable design generated from a linear block code that satisfies the CCP can be used in a coded caching scheme as follows. We associate the users with the blocks. Each subfile is associated with a point and an additional index. The placement scheme follows the natural incidence between the blocks and the points; a formal description is given in Algorithm 2 and illustrated further in Example 12.

Example 12. Consider the resolvable design from Example 10, where we recall that $z = 3$. The blocks in $\mathcal{A}$ correspond to twelve users U_{012}, U_{345}, U_{678}, U_{036}, U_{147}, U_{258}, U_{057}, U_{138}, U_{246}, U_{048}, U_{237}, U_{156}. Each file is partitioned into $F_s = 9 \times z = 27$ subfiles, each of which is denoted by $W_{n,t}^s$, $t = 0, \cdots, 8$, $s = 0, 1, 2$. The cache in user U_{abc}, denoted Z_{abc} is specified as $Z_{abc} = \{W_{n,t}^s \mid t \in$

```
Algorithm 2: Placement Scheme
  Input  : Resolvable design $(X, \mathcal{A})$ constructed from a $(n, k)$ linear block code. Let $z$ be
           the least positive integer such that $k + 1 \mid nz$.
  Divide each file $W_n$, for $n \in [N]$ into $q^k z$ subfiles. Thus,
    $W_n = \{W_{n,t}^s : t \in \{0, \dots, q^k - 1\}$ and $s \in \{0, \dots, z - 1\}\}$ ;
  User $U_B$ for $B \in \mathcal{A}$ caches $Z_B = \{W_{n,t}^s : n \in [N], t \in B$ and $s \in \{0, \dots, z - 1\}\}$ ;
  Output: Cache content of user $U_B$ denoted $Z_B$ for $B \in \mathcal{A}$.
```

$\{a, b, c\}, s \in \{0, 1, 2\}$ and $n \in [N]\}$. This corresponds to a coded caching system where each user caches 1/3-rd of each file so that $M/N = 1/3$.

In general, (see Algorithm 2) we have $K = |\mathcal{A}| = nq$ users. Each file W_n, $n \in [N]$ is divided into $q^k z$ subfiles $W_n = \{W_{n,t}^s \mid 0 \le t \le q^k - 1, 0 \le s \le z - 1\}$. A subfile $W_{n,t}^s$ is cached in user U_B where $B \in \mathcal{A}$ if $t \in B$. Therefore, each user caches a total of $Nq^{k-1}z$ subfiles. As each file consists of $q^k z$ subfiles, we have that $M/N = 1/q$.

It remains to show that we can design a delivery phase scheme that satisfies any possible demand pattern. Suppose that in the delivery phase user U_B requests file W_{d_B} where $d_B \in [N]$. The server responds by transmitting several equations that satisfy each user. Each equation allows $k+1$ users from *different parallel classes* to simultaneously obtain a missing subfile. Our delivery scheme is such that the set of transmitted equations can be classified into various *recovery sets* that correspond to appropriate collections of parallel classes. For example, in Fig. 3.1, $\mathcal{P}_{\mathcal{S}_0} = \{\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2\}, \mathcal{P}_{\mathcal{S}_1} = \{\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_3\}$ and so on. It turns out that these recovery sets correspond precisely to the sets $\mathcal{S}_a, 0 \le a \le \frac{zn}{k+1} - 1$ defined earlier. We illustrate this by means of the example below.

Example 13. Consider the placement scheme specified in Example 12. Let each user U_B request file W_{d_B}. The recovery sets are specified by means of the recovery set bipartite graph shown in Fig. 3.1, e.g., $\mathcal{P}_{\mathcal{S}_1}$ corresponds to $\mathcal{S}_1 = \{0, 1, 3\}$. The outgoing edges from each parallel class are labeled arbitrarily with numbers $0, 1$ and 2. Our delivery scheme is such that each user recovers missing subfiles with a specific superscript from each recovery set that its corresponding parallel class participates in. For instance, a user in parallel class $\mathcal{P}_1$ recovers missing subfiles with superscript 0

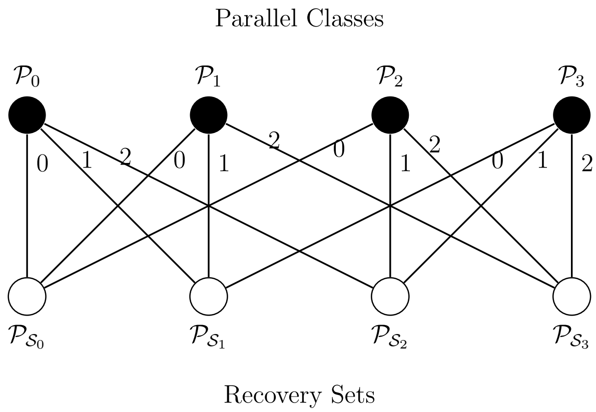
Parallel Classes
$\mathcal{P}_0$
$\mathcal{P}_1$
$\mathcal{P}_2$
$\mathcal{P}_3$
0
1
2
0
1
2
0
1
2
0
1
2
$\mathcal{P}_{\mathcal{S}_0}$
$\mathcal{P}_{\mathcal{S}_1}$
$\mathcal{P}_{\mathcal{S}_2}$
$\mathcal{P}_{\mathcal{S}_3}$
Recovery Sets

Figure 3.1 Recovery set bipartite graph

from $\mathcal{P}_{\mathcal{S}_0}$, superscript 1 from $\mathcal{P}_{\mathcal{S}_1}$ and superscript 2 from $\mathcal{P}_{\mathcal{S}_3}$; these superscripts are the labels of outgoing edges from $\mathcal{P}_1$ in the bipartite graph.

It can be verified, e.g., that user U_{012} which lies in $\mathcal{P}_0$ recovers all missing subfiles with superscript 1 from the equations below.

$$\begin{aligned} &W^1_{d_{012},3} \oplus W^1_{d_{036},2} \oplus W^0_{d_{237},0}, \qquad && W^1_{d_{012},6} \oplus W^1_{d_{036},1} \oplus W^0_{d_{156},0}, \\ &W^1_{d_{012},4} \oplus W^1_{d_{147},0} \oplus W^0_{d_{048},1}, && W^1_{d_{012},7} \oplus W^1_{d_{147},2} \oplus W^0_{d_{237},1}, \\ &W^1_{d_{012},8} \oplus W^1_{d_{258},0} \oplus W^0_{d_{048},2}, && W^1_{d_{012},5} \oplus W^1_{d_{258},1} \oplus W^0_{d_{156},2}. \end{aligned}$$

Each of the equations above benefits three users. They are generated simply by choosing U_{012} from $\mathcal{P}_0$, *any* block from $\mathcal{P}_1$ and the last block from $\mathcal{P}_3$ so that the *intersection of all these blocks is empty.* The fact that these equations are useful for the problem at hand is a consequence of the CCP. The process of generating these equations can be applied to all possible recovery sets. It can be shown that this allows all users to be satisfied at the end of the procedure.

In what follows, we first show that for the recovery set $\mathcal{P}_{\mathcal{S}_a}$ it is possible to generate equations that benefit $k+1$ users simultaneously.

Claim 1. Consider the resolvable design $(X, \mathcal{A})$ constructed as described in Section III.A by a (n, k) linear block code that satisfies the CCP. Let $\mathcal{P}_{\mathcal{S}_a} = \{\mathcal{P}_i \mid i \in \mathcal{S}_a\}$ for $0 \leq a \leq \frac{zn}{k+1} - 1$, i.e., it is the subset of parallel classes corresponding to $\mathcal{S}_a$. We emphasize that $|\mathcal{P}_{\mathcal{S}_a}| = k+1$. Consider blocks $B_{i_1,l_{i_1}}, \ldots, B_{i_k,l_{i_k}}$ (where $l_{i_j} \in \{0, \ldots, q-1\}$) that are picked from any k distinct parallel classes of $\mathcal{P}_{\mathcal{S}_a}$. Then, $|\cap_{j=1}^{k} B_{i_j,l_{i_j}}| = 1$.

Before proving Claim 1, we discuss its application in the delivery phase. Note that the claim asserts that k blocks chosen from k distinct parallel classes intersect in precisely one point. Now, suppose that one picks $k+1$ users from $k+1$ distinct parallel classes, such that their intersection is empty. These blocks (equivalently, users) can participate in an equation that benefits $k+1$ users. In particular, each user will recover a missing subfile indexed by the intersection of the other k blocks. We emphasize here that Claim 1 is at the core of our delivery phase. Of course, we need to justify that enough equations can be found that allow all users to recover all their missing subfiles. This follows from a natural counting argument that is made more formally in the subsequent discussion. The superscripts $s \in \{0, \ldots, z-1\}$ are needed for the counting argument to go through.

Proof. Following the construction in Section III.A, we note that a block $B_{i,l} \in \mathcal{P}_i$ is specified by

$$B_{i,l} = \{j : \mathbf{T}_{i,j} = l\}.$$

Now consider $B_{i_1,l_{i_1}}, \ldots, B_{i_k,l_{i_k}}$ (where $i_j \in \mathcal{S}_a, l_{i_j} \in \{0, \ldots, q-1\}$) that are picked from k distinct parallel classes of $\mathcal{P}_{\mathcal{S}_a}$. W.l.o.g. we assume that $i_1 < i_2 < \cdots < i_k$. Let $\mathcal{I} = \{i_1, \ldots, i_k\}$ and $\mathbf{T}_{\mathcal{I}}$ denote the submatrix of $\mathbf{T}$ obtained by retaining the rows in $\mathcal{I}$. We will show that the vector $[l_{i_1} \; l_{i_2} \; \ldots \; l_{i_k}]^T$ is a column in $\mathbf{T}_{\mathcal{I}}$ and only appears once.

To see this consider the system of equations in variables $\mathbf{u}_0, \ldots, \mathbf{u}_{k-1}$.

$$\sum_{b=0}^{k-1} \mathbf{u}_b g_{bi_1} = l_{i_1},$$

$$\vdots$$

$$\sum_{b=0}^{k-1} \mathbf{u}_b g_{bi_k} = l_{i_k}.$$

By the CCP, the vectors $\mathbf{g}_{i_1}, \mathbf{g}_{i_2}, \ldots, \mathbf{g}_{i_k}$ are linearly independent. Therefore this system of k equations in k variables has a unique solution over $GF(q)$. The result follows. □

```
Algorithm 3: Signal Generation Algorithm for $\mathcal{P}_{\mathcal{S}_a}$
Input : For $\mathcal{P} \in \mathcal{P}_{\mathcal{S}_a}$, $E(\mathcal{P}) = \text{label}(\mathcal{P} - \mathcal{P}_{\mathcal{S}_a})$. Signal set $Sig = \emptyset$.
while any user $U_B \in \mathcal{P}_j, j \in \mathcal{S}_a$ does not recover all its missing subfiles with superscript $E(\mathcal{P}_j)$ do
    Pick blocks $B_{j,l_j} \in \mathcal{P}_j$ for all $j \in \mathcal{S}_a$ and $l_j \in \{0, \dots, q-1\}$ such that $\cap_{j \in \mathcal{S}_a} B_{j,l_j} = \emptyset$;
    /* Pick blocks from distinct parallel classes in $\mathcal{P}_{\mathcal{S}_a}$ such that their intersection is
       empty */
    Let $\hat{l}_s = \cap_{j \in \mathcal{S}_a \setminus \{s\}} B_{j,l_j}$ for $s \in \mathcal{S}_a$;
    /* Determine the missing subfile index that the user from $\mathcal{P}_s$ will recover */
    Add signal $\oplus_{s \in S_a} W^{E(\mathcal{P}_s)}_{\kappa_{s,l_s},\hat{l}_s}$ to $Sig$ /* User $U_{B_{s,l_s}}$ demands file $W_{\kappa_{s,l_s}}$. This equation
       allows it to recover the corresponding missing subfile index $\hat{l}_s$. The superscript is
       determined by the recovery set bipartite graph */
end
Output: Signal set $Sig$.
```

We now provide an intuitive argument for the delivery phase. Recall that we form a recovery set bipartite graph (see Fig. 3.1 for an example) with parallel classes and recovery sets as the disjoint vertex subsets. The edges incident on each parallel class are labeled arbitrarily from $0, \dots, z-1$. For a parallel class $\mathcal{P} \in \mathcal{P}_{\mathcal{S}_a}$ we denote this label by $\text{label}(\mathcal{P} - \mathcal{P}_{\mathcal{S}_a})$. For a given recovery set $\mathcal{P}_{\mathcal{S}_a}$, the delivery phase proceeds by choosing blocks from distinct parallel classes in $\mathcal{P}_{\mathcal{S}_a}$ such that their intersection is empty; this provides an equation that benefits $k+1$ users. It turns out that the equation allows a user in parallel class $\mathcal{P} \in \mathcal{P}_{\mathcal{S}_a}$ to recover a missing subfile with the superscript $\text{label}(\mathcal{P} - \mathcal{P}_{\mathcal{S}_a})$.

The formal argument is made in Algorithm 3. For ease of notation in Algorithm 3, we denote the demand of user $U_{B_{i,j}}$ for $0 \le i \le n-1, 0 \le j \le q-1$ by $W_{\kappa_{i,j}}$.

Claim 2. Consider a user U_B belonging to parallel class $\mathcal{P} \in \mathcal{P}_{\mathcal{S}_a}$. The signals generated in Algorithm 3 can recover all the missing subfiles needed by U_B with superscript $E(\mathcal{P})$.

Proof. Let $\mathcal{P}_\alpha \in \mathcal{P}_{\mathcal{S}_a}$. In the arguments below, we argue that user $U_{B_{\alpha,l_\alpha}}$ that demands file $W_{\kappa_{\alpha,l_\alpha}}$ can recover all its missing subfiles with superscript $E(\mathcal{P}_\alpha)$. Note that $|B_{\alpha,l_\alpha}| = q^{k-1}$. Thus, user $U_{B_{\alpha,l_\alpha}}$ needs to obtain $q^k - q^{k-1}$ missing subfiles with superscript $E(\mathcal{P}_\alpha)$. Consider an iteration of the while loop where block B_{α,l_α} is picked in step 2. The equation in Algorithm 3 allows it to recover $W^{E(\mathcal{P}_\alpha)}_{\kappa_{\alpha,l_\alpha},\hat{l}_\alpha}$ where $\hat{l}_\alpha = \cap_{j\in\mathcal{S}_a\setminus\{\alpha\}} B_{j,l_j}$. This is because $\cap_{j\in\mathcal{S}_a} B_{j,l_j} = \emptyset$ and Claim 1.

Next we count the number of equations that $U_{B_{\alpha,l_\alpha}}$ participates in. We can pick $k-1$ users from some $k-1$ distinct parallel classes in $\mathcal{P}_{\mathcal{S}_a}$. This can be done in q^{k-1} ways. Claim 1 ensures that the blocks so chosen intersect in a single point. Next we pick a block from the only remaining parallel class in $\mathcal{P}_{\mathcal{S}_a}$ such that the intersection of all blocks is empty. This can be done in $q-1$ ways. Thus, there are a total of $q^{k-1}(q-1) = q^k - q^{k-1}$ equations in which user $U_{B_{\alpha,l_\alpha}}$ participates in.

It remains to argue that each equation provides a distinct subfile. Towards this end, let $\{i_1,\dots,i_k\} \subset \mathcal{S}_a$ be an index set such that $\alpha \notin \{i_1,\dots,i_k\}$. Suppose that there exist sets of blocks $\{B_{i_1,l_{i_1}},\dots,B_{i_k,l_{i_k}}\}$ and $\{B_{i_1,l'_{i_1}},\dots,B_{i_k,l'_{i_k}}\}$ such that $\{B_{i_1,l_{i_1}},\dots,B_{i_k,l_{i_k}}\} \neq \{B_{i_1,l'_{i_1}},\dots,B_{i_k,l'_{i_k}}\}$, but $\cap_{j=1}^k B_{i_j,l_{i_j}} = \cap_{j=1}^k B_{i_j,l'_{i_j}} = \beta$. This is a contradiction since this in turn implies that $\beta \in \cap_{j=2}^{k+1} B_{i_j,l_{i_j}} \bigcap \cap_{j=2}^{k+1} B_{i_j,l'_{i_j}}$, which is impossible since two blocks from the same parallel class have an empty intersection.

As the algorithm is symmetric with respect to all blocks in parallel classes belonging to $\mathcal{P}_{\mathcal{S}_a}$, we have the required result. □

The overall delivery scheme repeatedly applies Algorithm 3 to each of the recovery sets.

Lemma 2. The proposed delivery scheme terminates and allows each user's demand to be satisfied. Furthermore the transmission rate of the server is $\frac{(q-1)n}{k+1}$ and the subpacketization level is $q^k z$.

Proof. See Appendix. □

The main requirement for Lemma 2 to hold is that the recovery set bipartite graph be biregular, where multiple edges between the same pair of nodes is disallowed and the degree of each parallel class is z. It is not too hard to see that this follows from the definition of the recovery sets (see the proof in the Appendix for details).

In an analogous manner, if one starts with the generator matrix of a code that satisfies the (k, α)-CCP for $\alpha \leq k$, then we can obtain the following result which is stated below. The details are similar to the discussion for the $(k, k+1)$-CCP and can be found in the Appendix (Section A).

Corollary 1. Consider a coded caching scheme obtained by forming the resolvable design obtained from a (n, k) code that satisfies the (k, α)-CCP where $\alpha \leq k$. Let z be the least positive integer such that $\alpha \mid nz$. Then, a delivery scheme can be constructed such that the transmission rate is $\frac{(q-1)n}{\alpha}$ and the subpacketization level is $q^k z$.

3.2.4 Obtaining a scheme for $M/N = 1 - \frac{k+1}{nq}$.

The construction above works for a system where $M/N = 1/q$. It turns out that this can be converted into a scheme for $\frac{M}{N} = 1 - \frac{k+1}{nq}$. Thus, any convex combination of these two points can be obtained by memory-sharing.

Towards this end, we note that the class of coded caching schemes considered here can be specified by an *equation-subfile* matrix. This is inspired by the hypergraph formulation and the placement delivery array (PDA) based schemes for coded caching in Shangguan et al. (2018) and Yan et al. (2017a). Each equation is assumed to be of the all-but-one type, i.e., it is of the form $W_{d_{t_1}, \mathcal{A}_{j_1}} \oplus W_{d_{t_2}, \mathcal{A}_{j_2}} \oplus \cdots \oplus W_{d_{t_m}, \mathcal{A}_{j_m}}$ where for each $\ell \in [m]$, we have the property that user U_{t_ℓ} does not cache subfile $W_{n, \mathcal{A}_{j_\ell}}$ but caches all subfiles $W_{n, \mathcal{A}_{j_s}}$ where $\{j_s : s \in [m], s \neq \ell\}$.

The coded caching system corresponds to a $\Delta \times F_s$ equation-subfile matrix $\mathbf{S}$ as follows. We associate each row of $\mathbf{S}$ with an equation and each column with a subfile. We denote the i-th row of $\mathbf{S}$ by Eq_i and j-th column of $\mathbf{S}$ by $\mathcal{A}_j$. The value $\mathbf{S}(i, j) = t$ if in the i-th equation, user U_t recovers subfile $W_{d_t, \mathcal{A}_j}$, otherwise, $\mathbf{S}(i, j) = 0$. Suppose that these Δ equations allow each user to satisfy their demands, i.e., $\mathbf{S}$ corresponds to a valid coded caching scheme. It is not too hard to see that the placement scheme can be obtained by examining $\mathbf{S}$. Namely, user U_t caches the subfile corresponding to the j-th column if integer t does not appear in the j-th column.

Example 14. Consider a coded caching system in Maddah-Ali and Niesen (2014b) with $K = 4$, $\Delta = 4$ and $F_s = 6$. We denote the four users as U_1, U_2, U_3, U_4. Suppose that $\mathbf{S}$ is

$$\begin{array}{c|cccccc} & \mathcal{A}_1 & \mathcal{A}_2 & \mathcal{A}_3 & \mathcal{A}_4 & \mathcal{A}_5 & \mathcal{A}_6 \\ \hline Eq_1 & 3 & 2 & 0 & 1 & 0 & 0 \\ Eq_2 & 4 & 0 & 2 & 0 & 1 & 0 \\ Eq_3 & 0 & 4 & 3 & 0 & 0 & 1 \\ Eq_4 & 0 & 0 & 0 & 4 & 3 & 2 \end{array}.$$

Upon examining $\mathbf{S}$ it is evident for instance that user U_1 caches subfiles $\mathcal{A}_1, \ldots, \mathcal{A}_3$ as the number 1 does not appear in the corresponding columns. Similarly, the cache placement of the other users can be obtained. Interpreting this placement scheme in terms of the user-subfile assignment, it can be verified that the design so obtained corresponds to the transpose of the scheme considered in Example 3.1 (and also to the scheme of Maddah-Ali and Niesen (2014b) for $K = 4$, $M/N = 1/2$).

Lemma 3. Consider a $\Delta \times F_s$ equation-subfile matrix $\mathbf{S}$ whose entries belong to the set $\{0, 1, \ldots, K\}$. It corresponds to a valid coded caching system if the following three conditions are satisfied.

- There is no non-zero integer appearing more than once in each column.
- There is no non-zero integer appearing more than once in each row.
- If $\mathbf{S}(i_1, j_1) = \mathbf{S}(i_2, j_2) \neq 0$, then $\mathbf{S}(i_1, j_2) = \mathbf{S}(i_2, j_1) = 0$.

Proof. The placement scheme is obtained as discussed earlier, i.e., user U_t caches subfiles $W_{n,\mathcal{A}_j}$ if integer t does not appear in column $\mathcal{A}_j$. Therefore, matrix $\mathbf{S}$ corresponds to a placement scheme.

Next we discuss the delivery scheme. Note that Eq_i corresponds to an equation as follows.

$$W_{d_{t_1}, \mathcal{A}_{j_1}} \oplus W_{d_{t_2}, \mathcal{A}_{j_2}} \oplus \cdots \oplus W_{d_{t_m}, \mathcal{A}_{j_m}},$$

where $\mathbf{S}(i, j_1) = t_1, \cdots, \mathbf{S}(i, j_m) = t_m$. The above equation can allow m users to recover subfiles simultaneously if (a) U_{t_ℓ} does not cache $W_{n,\mathcal{A}_{j_\ell}}$ and (b) U_{t_ℓ} caches all $W_{n,\mathcal{A}_{j_s}}$ where $\{j_s : s \in [m], s \neq \ell\}$. It is evident that U_{t_ℓ} does not cache $W_{n,\mathcal{A}_{j_\ell}}$ owing to the placement scheme. Next, to guarantee the condition (b), we need to show that integer $t_\ell = \mathbf{S}(i, j_\ell)$ will not appear in column $\mathcal{A}_{j_s}$ in $\mathbf{S}$ where $\{j_s : s \in [m], s \neq \ell\}$. Towards this end, $t_\ell \neq \mathbf{S}(i, j_s)$ because of Condition 2. Next, consider the non-zero entries that lie in the column $\mathcal{A}_{j_s}$ but not in the row Eq_i. Assume there exists an entry $\mathbf{S}(i', j_s)$ such that $\mathbf{S}(i', j_s) = \mathbf{S}(i, j_\ell) = t_\ell$ and $i' \neq i$, then $\mathbf{S}(i, j_s) = t_s \neq 0$, which is a contradiction to Condition 3. Finally, Condition 1 guarantees that each missing subfile is recovered only once. □

User U_t caches a fraction $\frac{M_t}{N} = \frac{L_t}{F_s}$ where L_t is the number of columns of $\mathbf{S}$ that do not have the entry t. Similarly, the transmission rate is given by $R = \frac{\Delta}{F_s}$.

The crucial point is that the transpose of $\mathbf{S}$, i.e., $\mathbf{S}^T$ also corresponds to a coded caching scheme. This follows directly from the fact that $\mathbf{S}^T$ also satisfies the conditions in Lemma 3. In particular, $\mathbf{S}^T$ corresponds to a coded caching system with K users and Δ subfiles. In the placement phase, the cache size of U_t is $\frac{M_t'}{N} = \frac{\Delta - F_s + L_t}{\Delta}$. In the delivery phase, by transmitting F_s equations corresponding to the rows of $\mathbf{S}^T$, all missing subfiles can be recovered. Then, the transmission rate is $R = \frac{F_s}{\Delta}$.

Applying the above discussion in our context, consider the equation-subfile matrix $\mathbf{S}$ corresponding to the coded caching system with $K = nq$, $\frac{M_t}{N} = \frac{1}{q}$ for $1 \leq t \leq nq$, $F_s = q^k z$ and $\Delta = q^k(q-1)\frac{nz}{k+1}$. Then $\mathbf{S}^T$ corresponds to a system with $K' = nq$, $\frac{M'}{N} = 1 - \frac{k+1}{nq}$, $F_s' = (q-1)q^k\frac{zn}{k+1}$, and transmission rate $R' = \frac{F_s}{\Delta} = \frac{k+1}{(q-1)n}$. The following theorem is the main result of this paper.

Theorem 2. Consider a (n, k) linear block code over $GF(q)$ that satisfies the $(k, k+1)$ CCP. This corresponds to a coded caching scheme with $K = nq$ users, N files in the server where each user has a cache of size $M \in \left\{\frac{1}{q}N, \left(1 - \frac{k+1}{nq}\right)N\right\}$. Let z be the least positive integer such that $k+1 \mid nz$. When $\frac{M}{N} = \frac{1}{q}$, we have

$$R = \frac{(q-1)n}{k+1}, \text{ and}$$

$$F_s = q^k z.$$

When $\frac{M}{N} = (1 - \frac{k+1}{nq})$, we have

$$R = \frac{k+1}{(q-1)n}, \text{ and}$$
$$F_s = (q-1)q^k \frac{zn}{k+1}.$$

By memory sharing any convex combination of these points is achievable.

In a similar manner for the (n, k) linear block code that satisfies the (k, α)-CCP over $GF(q)$, the caching system where $M/N = 1/q$ can be converted into a system where $K' = nq$, $\frac{M'}{N'} = 1 - \frac{\alpha}{nq}$, $F'_s = (q-1)q^k \frac{zn}{\alpha}$ and $R' = \frac{\alpha}{(q-1)n}$ using the equation-subfile technique. The arguments presented above apply with essentially no change.

3.3 Some classes of linear codes that satisfy the CCP

At this point we have established that linear block codes that satisfy the CCP are attractive candidates for usage in coded caching. In this section, we demonstrate that there are a large class of generator matrices that satisfy the CCP. For most of the section we work with matrices over a finite field of order q. In the last subsection, we discuss some constructions for matrices over $\mathbb{Z}$ mod q when q is not a prime or prime power. We summarize the constructions presented in this section in Table 3.1.

3.3.1 Maximum-distance-separable (MDS) codes

(n, k)-MDS codes with minimum distance $n - k + 1$ are clearly a class of codes that satisfy the CCP. In fact, for these codes any k columns of the generator matrix can be shown to be full rank. Note however, that MDS codes typically need large field size, e.g., $q + 1 \geq n$ (assuming that the MDS conjecture is true)Roth (2006). In our construction, the value of $M/N = 1/q$ and the number of users is $K = nq$. Thus, for large n, we will only obtain systems with small values of M/N, or equivalently large values of M/N (by Theorem 2 above). This may be restrictive in practice.

3.3.2 Cyclic Codes

A cyclic code is a linear block code, where the circular shift of each codeword is also a codeword Lin and Costello (2004). A (n, k) cyclic code over $GF(q)$ is specified by a monic polynomial $g(X) = \sum_{i=0}^{n-k} g_i X^i$ with coefficients from $GF(q)$ where $g_{n-k} = 1$ and $g_0 \neq 0$; $g(X)$ needs to divide the polynomial $X^n - 1$. The generator matrix of the cyclic code is obtained as below.

$$\mathbf{G} = \begin{bmatrix} g_0 & g_1 & \cdot & \cdot & \cdot & g_{n-k} & 0 & \cdot & \cdot & 0 \\ 0 & g_0 & g_1 & \cdot & \cdot & \cdot & g_{n-k} & 0 & \cdot & 0 \\ \vdots & & & & & & & & & \vdots \\ 0 & 0 & \cdot & 0 & g_0 & g_1 & \cdot & \cdot & \cdot & g_{n-k} \end{bmatrix}.$$

The following claim shows that for verifying the CCP for a cyclic code it suffices to pick *any* set of $k+1$ consecutive columns.

Claim 3. Consider a (n, k) cyclic code with generator matrix $\mathbf{G}$. Let $\mathbf{G}_{\mathcal{S}}$ denote a set of $k+1$ consecutive columns of $\mathbf{G}$. If each $k \times k$ submatrix of $\mathbf{G}_{\mathcal{S}}$ is full rank, then $\mathbf{G}$ satisfies the $(k, k+1)$-CCP.

Proof. Let the generator polynomial of the cyclic code be $\mathbf{g}(X)$, where we note that $\mathbf{g}(X)$ has degree $n-k$. Let $\mathbf{G}_{\mathcal{S}} = [\mathbf{g}_{(a)_n}, \mathbf{g}_{(a+1)_n}, \cdots, \mathbf{g}_{(a+k)_n}]$ where we assume that $\mathbf{G}_{\mathcal{S}}$ satisfies the $(k, k+1)$-CCP. Let

$$\begin{aligned} \mathbf{G}_{\mathcal{S}\setminus j} =& [\mathbf{g}_{(a)_n}, \ldots, \mathbf{g}_{(a+j-1)_n}, \mathbf{g}_{(a+j+1)_n}, \ldots, \mathbf{g}_{(a+k)_n}], \text{ and} \\ \mathbf{G}_{\mathcal{S}'\setminus j} =& [\mathbf{g}_{(a+i)_n}, \ldots, \mathbf{g}_{(a+j-1+i)_n}, \\ & \mathbf{g}_{(a+j+1+i)_n}, \ldots, \mathbf{g}_{(a+k+i)_n}]. \end{aligned}$$

We need to show that if $\mathbf{G}_{\mathcal{S}\setminus j}$ has full rank, then $\mathbf{G}_{\mathcal{S}'\setminus j}$ has full rank, for any $0 \leq j \leq k$.

As $\mathbf{G}_{\mathcal{S}\setminus j}$ has full rank, there is no codeword $\mathbf{c} \neq \mathbf{0}$ such that $\mathbf{c}((a)_n) = \cdots = \mathbf{c}((a+j-1)_n) = \mathbf{c}((a+j+1)_n) = \cdots = \mathbf{c}((a+k)_n) = 0$. By the definition of a cyclic code, any circular shift of a codeword results in another codeword that belongs to the code. Therefore, there is no codeword $\mathbf{c}'$ such that $\mathbf{c}'((a+i)_n) = \cdots = \mathbf{c}'(a+j-1+i)_n) = \mathbf{c}'((a+j+1+i)_n) = \cdots = \mathbf{c}'((a+k+i)_n) = 0$. Thus, $\mathbf{G}_{\mathcal{S}'\setminus j}$ has full rank. □

Claim 3 implies a low complexity search algorithm to determine if a cyclic code satisfies the CCP. Instead of checking all $\mathbf{G}_{\mathcal{S}_a}$, $0 \leq a \leq \frac{zn}{k+1} - 1$, in Definition 5, we only need to check an arbitrary $\mathbf{G}_{\mathcal{S}} = [\mathbf{g}_{(i)_n}, \mathbf{g}_{(i+1)_n}, \cdots, \mathbf{g}_{(i+k)_n}]$, for $0 \leq i < n$. To further simplify the search, we choose $i = n - \lfloor \frac{k}{2} \rfloor - 1$.

For this choice of i, Claim 4 shows that $\mathbf{G}_{\mathcal{S}}$ is such that we only need to check the rank of a list of small-dimension matrices to determine if each $k \times k$ submatrix of $\mathbf{G}_{\mathcal{S}}$ is full rank (the proof appears in the Appendix).

Claim 4. A cyclic code with generator matrix $\mathbf{G}$ satisfies the CCP if the following conditions hold.

- For $0 < j \leq \lfloor \frac{k}{2} \rfloor$, the submatrices

$$\mathbf{C}_j = \begin{bmatrix} g_{n-k-1} & g_{n-k} & 0 & \cdot & \cdot & 0 \\ g_{n-k-2} & g_{n-k-1} & g_{n-k} & 0 & \cdot & 0 \\ \vdots & & & & & \vdots \\ g_{n-k-j+1} & \cdot & \cdot & \cdot & \cdot & g_{n-k} \\ g_{n-k-j} & \cdot & \cdot & \cdot & \cdot & g_{n-k-1} \end{bmatrix}$$

  have full rank. In the above expression, $g_i = 0$ if $i < 0$.

- For $\lfloor \frac{k}{2} \rfloor < j < k$, the submatrices

$$\mathbf{C}_j = \begin{bmatrix} g_1 & g_2 & \cdot & \cdot & \cdot & \cdot & g_{k-j} \\ g_0 & g_1 & \cdot & \cdot & \cdot & \cdot & g_{k-j-1} \\ \vdots & & & & & & \vdots \\ 0 & \cdot & \cdot & 0 & g_0 & g_1 & g_2 \\ 0 & \cdot & & \cdot\cdot & 0 & g_0 & g_1 \end{bmatrix}$$

  have full rank.

Example 15. Consider the polynomial $\mathbf{g}(X) = X^4 + X^3 + X + 2$ over $GF(3)$. Since it divides $X^8 - 1$, it is the generator polynomial of a $(8, 4)$ cyclic code over $GF(3)$. The generator matrix of this code is given below.

$$\mathbf{G} = \begin{bmatrix} 2 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 2 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

It can be verified that the 4×5 submatrix which consists of the two leftmost columns and three rightmost columns of $\mathbf{G}$ is such that all 4×4 submatrices of it are full rank. Thus, by Claim 3 the (4,5)-CCP is satisfied for $\mathbf{G}$.

Remark 3. Cyclic codes form an important class of codes that satisfy the (k, k)-CCP (*cf.* Definition 6). This is because, it is well-known Lin and Costello (2004) that any k consecutive columns of the generator matrix of a cyclic code are linearly independent.

3.3.3 Constructions leveraging properties of smaller base matrices

It is well recognized that cyclic codes do not necessarily exist for any choice of parameters. This is because of the divisibility requirement on the generator polynomial. We now discuss a more general construction of generator matrices that satisfy the CCP. As we shall see, this construction provides a more or less satisfactory solution for a large range of system parameters.

Our first simple observation is that the Kronecker product (denoted by $\otimes$ below) of a $z \times \alpha$ generator matrix that satisfies the (z, z)-CCP with a $t \times t$ identity matrix, $\mathbf{I}_{t \times t}$ immediately yields a generator matrix that satisfies the (tz, tz)-CCP.

Claim 5. Consider a (n, k) linear block code over $GF(q)$ whose generator matrix is specified as $\mathbf{G} = \mathbf{A} \otimes \mathbf{I}_{t \times t}$ where $\mathbf{A}$ is a $z \times \alpha$ matrix that satisfies the (z, z)-CCP. Then, $\mathbf{G}$ satisfies the (k, k)-CCP where $k = tz$ and $n = t\alpha$.

Proof. The recovery set for $\mathbf{A}$ is specified as $\mathcal{S}_a^z = \{(az)_\alpha, \cdots, (az + z - 1)_\alpha\}$ and the recovery set for $\mathbf{G}$ is specified as $\mathcal{S}_a^k = \{(ak)_n, \cdots, (ak + k - 1)_n\}$. Since $\mathbf{A}$ satisfies the (z, z)-CCP, $\mathbf{A}_{\mathcal{S}_a^z}$ has full rank. Note that $\mathbf{G}_{\mathcal{S}_a^k} = \mathbf{A}_{\mathcal{S}_a^z} \otimes \mathbf{I}_{t \times t}$. Then $\det(\mathbf{G}_{\mathcal{S}_a^k}) = \det(\mathbf{A}_{\mathcal{S}_a^z} \otimes \mathbf{I}_{t \times t}) = \det(\mathbf{A}_{\mathcal{S}_a^z})^t \neq 0$. Therefore, $\mathbf{G}$ satisfies the (k, k)-CCP. □

Remark 4. Let $\mathbf{A}$ be the generator matrix of a cyclic code over $GF(q)$, then $\mathbf{G} = \mathbf{A} \otimes \mathbf{I}_{t\times t}$ satisfies the (k, k)-CCP by Claim 5.

Our next construction addresses the $(k, k+1)$-CCP. In what follows, we use the following notation.

- $\mathbf{1}_a$: $\underbrace{[1, \cdots, 1]}_{a}^T$;
- $\mathbf{C}(c_1, c_2)_{a\times b}$: $a \times b$ matrix where each row is the cyclic shift (one place to the right) of the row above it and the first row is $[c_1 \; c_2 \; 0 \; \cdots \; 0]$; and
- $\mathbf{0}_{a\times b}$: $a \times b$ matrix with zero entries.

Consider parameters n, k. Let the greatest common divisor of n and $k+1$, $\gcd(n, k+1) = t$. It is easy to verify that $z = \frac{k+1}{t}$ is the smallest integer such that $k+1 \mid nz$. Let $n = t\alpha$ and $k+1 = tz$. Claim 6 below constructs a (n, k) linear code that satisfies the CCP over $GF(q)$ where $q > \alpha$. Since $\alpha = \frac{n}{t}$, the required field size in Claim 6 is lower than the MDS code considered in Section 3.3.1.

Claim 6. Consider a (n, k) linear block code over $GF(q)$ whose generator matrix is specified as follows,

$$\mathbf{G} = \begin{bmatrix} b_{00}\mathbf{I}_{t\times t} & \cdots & b_{0(\alpha-1)}\mathbf{I}_{t\times t} \\ b_{10}\mathbf{I}_{t\times t} & \cdots & b_{1(\alpha-1)}\mathbf{I}_{t\times t} \\ \vdots & & \vdots \\ b_{(z-2)0}\mathbf{I}_{t\times t} & \cdots & b_{(z-2)(\alpha-1)}\mathbf{I}_{t\times t} \\ \mathbf{C}(b_{(z-1)0}, b_{(z-1)0})_{(t-1)\times t} & \cdots & \mathbf{C}(b_{(z-1)(\alpha-1)}, b_{(z-1)(\alpha-1)})_{(t-1)\times t} \end{bmatrix} \tag{3.3}$$

where

$$\begin{bmatrix} b_{00} & b_{01} & \cdots & b_{0(\alpha-1)} \\ b_{10} & b_{11} & \cdots & b_{1(\alpha-1)} \\ \vdots & & & \vdots \\ b_{(z-1)0} & b_{(z-1)1} & \cdots & b_{(z-1)(\alpha-1)} \end{bmatrix}$$

is a Vandermonde matrix and $q > \alpha$. Then, $\mathbf{G}$ satisfies the $(k, k+1)$-CCP.

Proof. The proof again leverages the idea that $\mathbf{G}$ can be expressed succinctly by using Kronecker products. The arguments can be found in the Appendix. □

Consider the case when $\alpha = z + 1$. We construct a (n, k) linear code satisfy the CCP over $GF(q)$ where $q \geq z$. It can be noted that the constraint of field size is looser than the corresponding constraint in Claim 6.

Claim 7. Consider $nz = (z + 1) \cdot (k + 1)$. Consider a (n, k) linear block code whose generator matrix (over $GF(q)$) is specified as follows.

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_{t\times t} & \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times t} & \mathbf{0}_{t\times(t-1)} & \mathbf{1}_t & b_1\mathbf{I}_{t\times t} \\ \mathbf{0}_{t\times t} & \mathbf{I}_{t\times t} & \cdots & \mathbf{0}_{t\times t} & \mathbf{0}_{t\times(t-1)} & \mathbf{1}_t & b_2\mathbf{I}_{t\times t} \\ \vdots & & & & & & \vdots \\ \mathbf{0}_{t\times t} & \mathbf{0}_{t\times t} & \cdots & \mathbf{I}_{t\times t} & \mathbf{0}_{t\times(t-1)} & \mathbf{1}_t & b_{z-1}\mathbf{I}_{t\times t} \\ \mathbf{0}_{(t-1)\times t} & \mathbf{0}_{(t-1)\times t} & \cdots & \mathbf{0}_{(t-1)\times t} & \mathbf{I}_{(t-1)\times(t-1)} & \mathbf{1}_{t-1} & \mathbf{C}(c_1,c_2)_{(t-1)\times t}, \end{bmatrix} \tag{3.4}$$

where $t = n - k - 1$. If $q \geq z$, $b_1, b_2, \cdots, b_{z-1}$ are non-zero and distinct, and $c_1 + c_2 = 0$, then $\mathbf{G}$ satisfies the CCP.

Proof. See Appendix. □

Given a (n, k) code that satisfies the CCP, we can use it obtain higher values of n in a simple manner as discussed in the claim below.

Claim 8. Consider a (n, k) linear block code over $GF(q)$ with generator matrix $\mathbf{G}$ that satisfies the CCP. Let the first $k+1$ columns of $\mathbf{G}$ be denoted by the submatrix $\mathbf{D}$. Then the matrix $\mathbf{G}'$ of dimension $k \times (n + s(k + 1))$ where $s \geq 0$

$$\mathbf{G}' = [\underbrace{\mathbf{D}|\cdots|\mathbf{D}}_{s}|\mathbf{G}]$$

also satisfies the CCP.

Proof. See Appendix. □

Claim 8 can provide more parameter choices and more possible code constructions. For example, given n, k, q, where $k+1+(n)_{k+1} \leq q+1 < n$, there may not exist a (n, k)-MDS code over $GF(q)$. However, there exists a $(k + 1 + (n)_{k+1}, k)$-MDS code over $GF(q)$. By Claim 8, we can obtain a (n, k) linear block code over $GF(q)$ that satisfies the CCP. Similarly, combining Claim 4, Claim 6, Claim 7 with Claim 8, we can obtain more linear block codes that satisfy the CCP.

A result very similar to Claim 8 can be obtained for the (k, α)-CCP. Specifically, consider a (n, k) linear block code with generator matrix $\mathbf{G}$ that satisfies the (k, α)-CCP and let $\mathbf{D}$ be the first α columns of $\mathbf{G}$. Then, $\mathbf{G}' = [\underbrace{\mathbf{D}|\cdots|\mathbf{D}}_{s}|\mathbf{G}]$ of dimension $k \times (n + s\alpha)$ also satisfies the (k, α)-CCP.

3.3.4 Constructions where q is not a prime or a prime power

We now discuss constructions where q is not a prime or a prime power. We attempt to construct matrices over the ring $\mathbb{Z} \mod q$ in this case. The issue is somewhat complicated by the fact that a square matrix over $\mathbb{Z} \mod q$ has linear independent rows if and only if its determinant is a unit in the ring Dummit and Foote (2003). In general, this fact makes it harder to obtain constructions such as those in Claim 6 that exploit the Vandermonde structure of the matrices. Specifically, the difference of units in a ring is not guaranteed to be a unit. However, we can still provide some constructions. It can be observed that Claim 5 and Claim 8 hold for linear block codes over $\mathbb{Z} \mod q$. We will use them without proof in this subsection.

Claim 9. Let $\mathbf{G} = [\mathbf{I}_{k\times k}|\mathbf{1}_k]$, i.e., it is the generator matrix of a $(k+1, k)$ single parity check (SPC) code, where the entries are from $\mathbb{Z} \mod q$. The $\mathbf{G}$ satisfies the $(k, k+1)$-CCP and the (k, k)-CCP. It can be used as base matrix for Claim 5.

Proof. It is not too hard to see that when $\mathbf{G} = [\mathbf{I}_{k\times k}|\mathbf{1}_k]$, any $k \times k$ submatrix of $\mathbf{G}$ has a determinant which is ± 1, i.e., it is a unit over $\mathbb{Z} \mod q$. Thus, the result holds in this case. □

Claim 10. The following matrix with entries from $\mathbb{Z} \mod q$ satisfies the $(k, k+1)$-CCP. Here $k = 2t - 1$ and $n = 3t$.

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_{t\times t} & \mathbf{0} & \mathbf{1}_t & \mathbf{I}_{t\times t} \\ \mathbf{0} & \mathbf{I}_{(t-1)\times(t-1)} & \mathbf{1}_t & \mathbf{C}(1,-1)_{(t-1)\times t} \end{bmatrix}.$$

Proof. This can be proved by following the arguments in the proof of Claim 7 while treating elements to be from $\mathbb{Z} \mod q$ and setting $z = 2$. We need to consider three different $k \times (k+1)$ submatrices for which we need to check the property. These correspond to simpler instances of the submatrices considered in Types I - III in the proof of Claim 7. In particular, the corresponding determinants will always be ± 1 which are units over $\mathbb{Z} \mod q$. □

Remark 5. We note that the general construction in Claim 7 can potentially fail in the case when the matrices are over $\mathbb{Z} \mod q$. This is because in one of the cases under consideration (specifically, Type III, Case 1), the determinant depends on the difference of the b_i values. The difference of units in $\mathbb{Z} \mod q$ is not guaranteed to be a unit, thus there is no guarantee that the determinant is a unit.

Remark 6. We can use Claim 8 to obtain higher values of n based on the above two classes of linear block codes over $\mathbb{Z} \mod q$.

While most constructions of cyclic codes are over $GF(q)$, there has been some work on constructing cyclic codes over $\mathbb{Z} \mod q$. Specifically, Blake (1972) provides a construction where $q = q_1 \times q_2 \cdots \times q_d$ and $q_i, i = 1, \dots, d$ are prime. We begin by outlining this construction. By the Chinese remainder theorem any element $\gamma \in \mathbb{Z} \mod q$ has a unique representation in terms of its residues modulo q_i, for $i = 1, \dots, d$. Let $\psi : \mathbb{Z} \mod q \to GF(q_1) \times \cdots \times GF(q_d)$ denote this map.

- Suppose that (n, k_i) cyclic codes over $GF(q_i)$ exist for all $i = 1, \dots, d$. Each individual code is denoted $\mathcal{C}^i$.

- Let $\mathcal{C}$ denote the code over $\mathbb{Z} \mod q$. Let $\mathbf{c}^{(i)} \in \mathcal{C}^i$ for $i = 1, \dots, d$. The codeword $\mathbf{c} \in \mathcal{C}$ is obtained as follows. The j-th component of $\mathbf{c}$, $\mathbf{c}_j = \psi^{-1}(\mathbf{c}_j^{(1)}, \dots, \mathbf{c}_j^{(d)})$

Therefore, there are $q_1^{k_1} q_2^{k_2} \cdots q_d^{k_d}$ codewords in $\mathcal{C}$. It is also evident that $\mathcal{C}$ is cyclic. As discussed in Section 3.2.1, we form the matrix $\mathbf{T}$ for the codewords in $\mathcal{C}$. It turns out that using $\mathbf{T}$ and the technique discussed in Section 3.2.1, we can obtain a resolvable design. Furthermore, the gain of the system in the delivery phase can be shown to be $k_{min} = \min\{k_1, k_2, \cdots, k_d\}$. We discuss these points in detail in the Appendix (Section 5).

3.4 Discussion and Comparison with Existing Schemes

3.4.1 Discussion

When the number of users is $K = nq$ and the cache fraction is $\frac{M}{N} = \frac{1}{q}$, we have shown in Theorem 2 that the gain $g = k + 1$ and $F_s = q^k z$. Therefore, both the gain and the subpacketization level increase with larger k. Thus, for our approach given a subpacketization budget F_s', the highest coded gain that can be obtained is denoted by $g_{max} = k_{max} + 1$ where k_{max} is the largest integer such that $q^{k_{max}} z \leq F_s'$ and there exists a (n, k_{max}) linear block code that satisfies the CCP.

For determining k_{max}, we have to characterize the collection of values of k such that there exists a (n, k) linear code satisfies the CCP over $GF(q)$ or $\mathbb{Z} \mod q$. We use our proposed constructions (MDS code, Claim 4, Claim 6, Claim 7, Claim 8, Claim 9, Claim 10) for this purpose. We call this collection $\mathcal{C}(n, q)$ and generate it in Algorithm 4. We note here that it is entirely possible that there are other linear block codes that fit the appropriate parameters and are outside the scope of our constructions. Thus, the list may not be exhaustive. In addition, we note that we only check for the $(k, k+1)$-CCP. Working with the (k, α)-CCP where $\alpha \leq k$ can provide more operating points.

Example 16. Consider a caching system with $K = nq = 12 \times 5 = 60$ users and cache fraction $\frac{M}{N} = \frac{1}{5}$. Suppose that the subpacketization budget is 1.5×10^6. By checking all $k < n$ we can construct $\mathcal{C}(n, q)$ (see Table 3.2). As a result, $\mathcal{C}(n, q) = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 11\}$. Then $k_{max} = 8$, $F_s \approx 1.17 \times 10^6$ and the maximal coded gain we can achieve is $g_{max} = 9$. By contrast, the scheme in Maddah-Ali and Niesen (2014b) can achieve coded gain $g = \frac{KM}{N} + 1 = 13$ but requires subpacketization level $F_s = \binom{K}{\frac{KM}{N}} \approx 1.4 \times 10^{12}$.

We can achieve almost the same rate by performing memory-sharing by using the scheme of Maddah-Ali and Niesen (2014b) in this example. In particular, we divide each file of size Ω into two smaller subfiles W_n^1 and W_n^2, where the size of W_n^1, $|W_n^1| = \frac{9}{10}\Omega$ and the size of W_n^2, $|W_n^2| = \frac{1}{10}\Omega$. The scheme of Maddah-Ali and Niesen (2014b) is then applied separately on W_n^1 and W_n^2 with $\frac{M_1}{N_1} = \frac{2}{15}$ (corresponding to W_n^1) and $\frac{M_2}{N_2} = \frac{13}{15}$ (corresponding to W_n^2). Thus, the overall cache fraction is $0.9 \times \frac{2}{15} + 0.1 \times \frac{13}{15} \approx \frac{1}{5}$. The overall coded gain of this scheme is $g \approx 9$. However, the

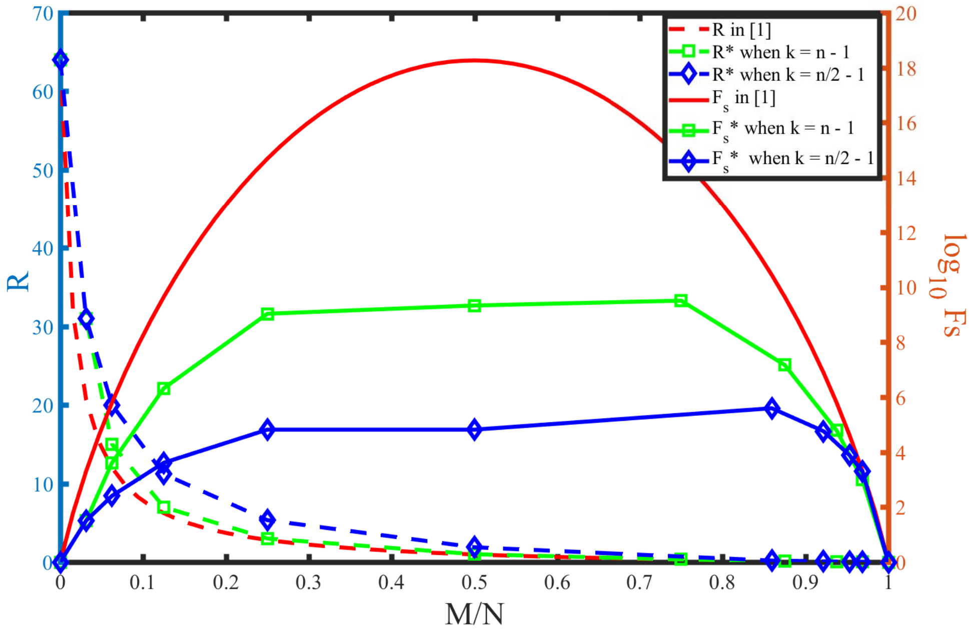

Figure 3.2 A comparison of rate and subpacketization level vs. M/N for a system with $K = 64$ users. The left y-axis shows the rate and the right y-axis shows the logarithm of the subpacketization level. The green and the blue curves correspond to two of our proposed constructions. Note that our schemes allow for multiple orders of magnitude reduction in subpacketization level and the expense of a small increase in coded caching rate.

subpacketization level is $F_s^{MN} = \binom{K}{KM_1/N_1} + \binom{K}{KM_2/N_2} \approx 5 \times 10^9$, which is much greater than the subpacketization budget.

In Fig. 3.2, we present another comparison for system parameters $K = 64$ and different values of M/N. The scheme of Maddah-Ali and Niesen (2014b) works for all M/N such that KM/N is an integer. In Fig. 3.2, our plots have markers corresponding to M/N values that our scheme achieves. For ease of presentation, both the rate (left y-axis) and the logarithm of the subpacketization level (right y-axis) are shown on the same plot. We present results corresponding to two of our construction techniques: (i) the SPC code and (ii) a smaller SPC code coupled with Claim 8. It can be seen that our subpacketization levels are several orders of magnitude smaller with only a small increase in the rate.

An in-depth comparison for general parameters is discussed next. In the discussion below, we shall use the superscript $*$ to refer to the rates and subpacketization levels of our proposed scheme.

3.4.2 Comparison with memory-sharing within the scheme of Maddah-Ali and Niesen (2014b)

Suppose that for given K, M and N, a given rate R can be achieved by the memory sharing of the scheme in Maddah-Ali and Niesen (2014b) between the corner points $(M_1, R_1), (M_2, R_2), \cdots, (M_d, R_d)$ where $M_i = \frac{t_i N}{K}$ for some integer t_i. Then $R_i = \frac{K(1-\frac{M_i}{N})}{1+\frac{KM_i}{N}}$, $R = \sum_{i=1}^{d} \lambda_i R_i$, $M/N = \sum_{i=1}^{d} \lambda_i \frac{M_i}{N}$ and $\sum_{i=1}^{d} \lambda_i = 1$. The subpacketization level is $F_s^{MS} = \sum_{i=1}^{d} \binom{K}{\frac{KM_i}{N}}$. In addition, we note that the function $h(x) = K(1-x)/(1+Kx)$ is convex in the parameter $0 \le x \le 1$. This can be verified by a simple second derivative calculation.

We first argue that F_s^{MS} is lower bounded by $\binom{K}{\frac{KM'}{N}}$, where M' is obtained as follows. For a given M/N, we first determine λ and $\frac{M^*}{N}$ that satisfy the following equations.

$$R = \lambda \frac{K(1-\frac{M^*}{N})}{1+\frac{KM^*}{N}} + (1-\lambda)\frac{K\frac{M^*}{N}}{1+K(1-\frac{M^*}{N})}, \text{ and} \tag{3.5}$$

$$\frac{M}{N} = \lambda \frac{M^*}{N} + (1-\lambda)\Big(1 - \frac{M^*}{N}\Big). \tag{3.6}$$

Here, $\frac{M^*}{N} \le \frac{1}{2}$, and $M' = \frac{t'N}{K}$, where t' is the least integer such that $M' \ge M^*$.

To see this, consider the following argument. Suppose that the above statement is not true. Then, there exists a scheme that operates via memory sharing between points $(M_1, R_1), \cdots, (M_d, R_d)$ such that $F_s < \binom{K}{K\frac{M'}{N}}$. Note that $\binom{K}{\frac{KM_1}{N}} < \binom{K}{\frac{KM_2}{N}}$ if $\frac{M_1}{N} < \frac{M_2}{N} \le \frac{1}{2}$ or $\frac{M_1}{N} > \frac{M_2}{N} \ge \frac{1}{2}$. By the convexity of $h(\cdot)$, we can conclude that (M, R) is not in the convex hull of the corner points $(M_1, R_1), \cdots, (M_d, R_d)$. This is a contradiction.

Next, we compare this lower bound on F_s^{MS} to the subpacketization level of our proposed scheme. In principle, we can solve the system of equations (3.5) and (3.6) for $R = \frac{n(q-1)}{k+1}$ and $\frac{M}{N} = \frac{1}{q}$ and obtain the appropriate λ and M^* values[1]. Unfortunately, doing this analytically becomes quite messy and does not yield much intuition. Instead, we illustrate the reduction in subpacketization level by numerical comparisons.

[1] Similar results can be obtained for $\frac{M}{N} = 1 - \frac{k+1}{nq}$

Algorithm 4: $\mathcal{C}(n,q)$ Construction Algorithm

```
Input   : $n$, $q$, $\mathcal{C}(n,q) = \emptyset$
if $q$ is a prime power then
    for $k = 1 : (n-1)$ do
        $n' \leftarrow (n)_{k+1} + k + 1$;
        $z \leftarrow \frac{k+1}{\gcd(n',k+1)}$;
        $\alpha \leftarrow \frac{n'}{\gcd(n',k+1)}$;
        if there exists a $(n' + i(k+1), k)$ cyclic code which satisfies the condition in Claim 4
         for some $i$ such that $n' + i(k+1) \leq n$ then
            $\mathcal{C}(n,q) \leftarrow k$. Corresponding codes are constructed by using Claim 8.
        else
            if $z \leq 2$ then
                $\mathcal{C}(n,q) \leftarrow k$. Corresponding codes are constructed by SPC code and Claim 8
                 when $z = 1$ or Claim 7 and Claim 8 when $z = 2$.
            else
                if $q + 1 \geq n'$ then
                    $\mathcal{C}(n,q) \leftarrow k$. Corresponding codes are constructed by MDS code and
                     Claim 8.
                else
                    if $\alpha = z + 1$ and $q \geq z$ then
                        $\mathcal{C}(n,q) \leftarrow k$. Corresponding codes are constructed by Claim 7 and
                         Claim 8.
                    end
                    if $\alpha > z + 1$ and $q > \alpha$ then
                        $\mathcal{C}(n,q) \leftarrow k$. Corresponding codes are constructed by Claim 6 and
                         Claim 8.
                    end
                end
            end
        end
    end
end
if $q$ is not a prime power then
    for $k = 1 : (n-1)$ do
        if $z \leq 2$ then
            $\mathcal{C}(n,q) \leftarrow k$. Corresponding codes are constructed by Claim 9 when $z = 1$ and
             Claim 8 or Claim 10 and Claim 8 when $z = 2$.
        end
    end
end
Output: $\mathcal{C}(n,q)$
```

Example 17. Consider a $(9,5)$ linear block code over $GF(2)$ with generator matrix specified below.

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

It can be checked that $\mathbf{G}$ satisfies the $(5,6)$-CCP. Thus, it corresponds to a coded caching system with $K = 9 \times 2 = 18$ users. Our scheme achieves the point $\frac{M_1}{N} = \frac{1}{2}$, $R_1 = \frac{3}{2}$, $F^*_{s,1} = 64$ and $\frac{M_2}{N} = \frac{2}{3}$, $R_2 = \frac{2}{3}$, $F^*_{s,2} = 96$.

On the other hand for $\frac{M_1}{N} = \frac{1}{2}$, $R_1 = \frac{3}{2}$, by numerically solving (3.5) and (3.6) we obtain $\frac{M_1^*}{N} \approx 0.227$ and therefore $\frac{M_1'}{N} = \frac{5}{18}$. Then $F_{s,1}^{MS} \geq \binom{18}{5} = 8568$, which is much higher than $F^*_{s,1} = 64$. A similar calculation shows that $\frac{M_2^*}{N} \approx \frac{1}{4}$ and therefore $\frac{M_2'}{N} = \frac{5}{18}$. Thus $F_{s,2}^{MS}$ is also at least as large as 8568, which is still much higher than $F^*_{s,2} = 96$.

The next set of comparisons are with other proposed schemes in the literature. We note here that several of these are restrictive in the parameters that they allow.

3.4.3 Comparison with Maddah-Ali and Niesen (2014b), Yan et al. (2017a), Yan et al. (2017b), Shangguan et al. (2018) and Shanmugam et al. (2017)

For comparison with Maddah-Ali and Niesen (2014b), denote R^{MN} and F_s^{MN} be the rate and the subpacketization level of the scheme of Maddah-Ali and Niesen (2014b), respectively. For the rate comparison, we note that

$$\begin{aligned} \frac{R^*}{R^{MN}} &= \frac{1+n}{1+k}, \quad \text{for } \frac{M}{N} = \frac{1}{q} \\ \frac{R^*}{R^{MN}} &= \frac{nq-k}{nq-n}, \quad \text{for } \frac{M}{N} = 1 - \frac{1+k}{nq}, \end{aligned}$$

For the comparison of subpacketization level we have the following results.

Claim 11. When $K = nq$, the following results hold.

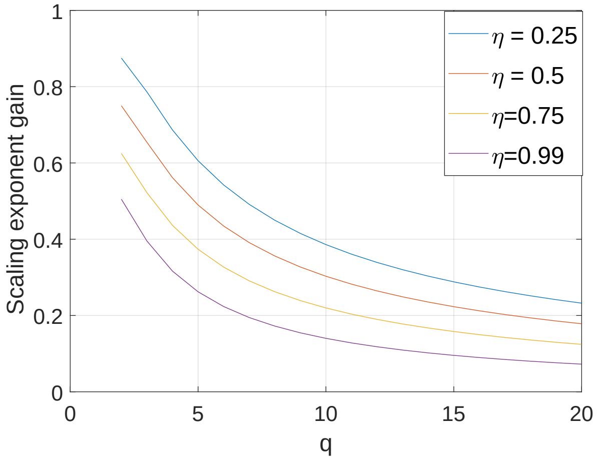

Figure 3.3 The plot shows the gain in the scaling exponent obtained using our techniques for different value of $M/N = 1/q$. Each curve corresponds to a choice of $\eta = k/n$.

- If $\frac{M}{N} = \frac{1}{q}$, we have

$$\lim_{n\to\infty} \frac{1}{K}\log_2 \frac{F_s^{MN}}{F_s^*} = H_2\left(\frac{1}{q}\right) - \frac{\eta}{q}\log_2 q. \tag{3.7}$$

- If $\frac{M}{N} = 1 - \frac{k+1}{nq}$, we have

$$\lim_{n\to\infty} \frac{1}{K}\log_2 \frac{F_s^{MN}}{F_s^*} = H_2\left(\frac{\eta}{q}\right) - \frac{\eta}{q}\log_2 q. \tag{3.8}$$

In the above expressions, $0 < \eta = k/n \le 1$ and $H_2(\cdot)$ represents the binary entropy function.

Proof. Both results are simple consequences of approximating $\binom{K}{Kp} \approx 2^{KH_2(p)}$ Graham et al. (1994). The derivations can be found in the Appendix. □

It is not too hard to see that F_s^* is exponentially lower than F_s^{MN}. Thus, our rate is higher, but the subpacketization level is exponentially lower. Thus, the gain in the scaling exponent of with respect to the scheme of Maddah-Ali and Niesen (2014b) depends on the choice of R and the value of M/N. In Fig. 3.3 we plot this value of different values of R and q. The plot assumes that codes satisfying the CCP can be found for these rates and corresponds to the gain in eq. (3.7).

In Yan et al. (2017a) a scheme for the case when $M/N = 1/q$ or $M/N = 1-1/q$ with subpacketization level exponentially smaller with respect to Maddah-Ali and Niesen (2014b) was presented.

This result can be recovered a special case of our work (Theorem 2) when the linear block code is chosen as a single parity check code over $\mathbb{Z} \mod q$. In this specific case, q does not need to be a prime power. Thus, our results subsume the results of Yan et al. (2017a).

In a more recent preprint, reference Yan et al. (2017b), proposed a caching system with $K = \binom{m}{a}$, $F_s = \binom{m}{b}$ and $M/N = 1 - \binom{a}{\lambda}\binom{m-a}{b-\beta}/\binom{m}{b}$. The corresponding rate is

$$R = \frac{\binom{m}{a+b-2\beta}}{\binom{m}{b}} \min\left\{ \binom{m-(a+b-2\beta)}{\lambda}, \binom{a+b-2\beta}{a-\beta} \right\},$$

where m, a, b, β are positive integers and $0 < a < m$, $0 < b < m$, $0 \leq \beta \leq \min\{a, b\}$. While a precise comparison is somewhat hard, we can compare the schemes for certain parameter choices, that were also considered in Yan et al. (2017b).

Let $a = 2$, $\beta = 1$, $m = 2b$. This corresponds to a coded caching system with $K = b(2b-1) \approx 2b^2$, $\frac{M}{N} = \frac{b-1}{2b-1} \approx \frac{1}{2}$, $F_s = \binom{2b}{b} \approx 2^{2b}$, $R = b$. For comparison with our scheme we keep the transmission rates of both schemes roughly the same and let $n = b^2$, $q = 2$, $k = b - 1$. We assume that the corresponding linear block code exists. Then $F_s^* \approx 2^b$, which is better than F_s.

On the other hand if we let $\beta = 0$, $a = 2$, $m = 2qb$, we obtain a coded caching system with $K = \frac{m(m-1)}{2}$, $\frac{M}{N} \approx \frac{1}{q}$, $F_s = \binom{m}{\frac{m}{2q}} \approx (2q)^{\frac{m}{2q}}$, $R^{YAN} = (2q-1)^2$. For keeping the rates the same, we let $n = \frac{m(m-1)}{2q}$, $k = \frac{m(m-1)}{4q(2q-1)} - 1$ so that $F_s^* \approx q^{\frac{m(m-1)}{4q(2q-1)}} \approx q^{\frac{m^2}{8q^2}}$. In this regime, the subpacketization level of Yan et al. (2017b) will typically be lower.

The work of Shangguan et al. (2018) proposed caching schemes with parameters (i) $K = \binom{m}{a}$, $\frac{M}{N} = 1 - \frac{\binom{m-a}{b}}{\binom{m}{b}}$, $F_s = \binom{m}{b}$ and $R = \frac{\binom{m}{a+b}}{\binom{m}{b}}$, where a, b, m are positive integers and $a + b \leq m$ and (ii) $K = \binom{m}{t}q^t$, $\frac{M}{N} = 1 - \frac{1}{q^t}$, $F_s = q^m(q-1)^t$ and $R = \frac{1}{(q-1)^t}$, where q, t, m are positive integers.

Their scheme (i), is the special case of scheme in Yan et al. (2017b) when $\beta = 0$. For the second scheme, if we let $t = 2$, Shangguan et al. (2018) shows that $R \approx R^*$, $F_s \approx q^{\sqrt{\frac{K}{2q}}}(\sqrt{q} - 1)^2$ and $F_s^* \approx (q-1)q^{\frac{K}{q}-1}$, which means F_s is again better than F_s^*. We emphasize here that these results require somewhat restrictive parameter settings.

Finally, we consider the work of Shanmugam et al. (2017). In their work, they leveraged the results of Alon et al. (2012) to arrive at coded caching schemes where the subpacketization is linear in K. Specifically, they show that for any constant M/N, there exists a scheme with rate K^δ, where

$\delta > 0$ can be chosen arbitrarily small by choosing K large enough. From a theoretical perspective, this is a positive result that indicates that regimes where linear subpacketization scaling is possible. However, these results are only valid when the value of K is very large. Specifically, $K = C^n$ and the result is asymptotic in the parameter n. For these parameter ranges, the result of Shanmugam et al. (2017) will clearly be better as compared to our work.

3.5 Conclusions and Future Work

In this work we have demonstrated a link between specific classes of linear block codes and the subpacketization problem in coded caching. Crucial to our approach is the consecutive column property which enforces that certain consecutive column sets of the corresponding generator matrices are full-rank. We present several constructions of such matrices that cover a large range of problem parameters. Leveraging this approach allows us to construct families of coded caching schemes where the subpacketization level is exponentially smaller compared to the approach of Maddah-Ali and Niesen (2014b).

There are several opportunities for future work. Even though our subpacketization level is significantly lower than Maddah-Ali and Niesen (2014b), it still scales exponentially with the number of users. Of course, the rate of growth with the number of users is much smaller. There have been some recent results on coded caching schemes that demonstrate the existence of schemes where the subpacketization scales sub-exponentially in the number of users. It would be interesting to investigate whether some of these ideas can be leveraged to obtain schemes that work for practical systems with tens or hundreds of users.

Table 3.1 A summary of the different constructions of CCP matrices in Section 3.3

Code type	**Code construction**	**Notes**
Codes over field $GF(q)$	(n,k) MDS codes	Satisfy $(k,k+1)$-CCP. Need $q+1 \geq n$.
	(n,k) Cyclic codes	Existence depends on certain properties of the generator polynomials. All cyclic codes satisfy the (k,k)-CCP. Need additional conditions for the $(k,k+1)$-CCP.
	Kronecker product of $z \times \alpha$ matrix satisfying the (z,z)-CCP with the identity matrix $\mathbf{I}_{t\times t}$	Satisfy the (k,k)-CCP where $k = tz$.
	Kronecker product of Vandermonde and Vandermonde-like matrices with structured base matrices	Satisfy the $(k,k+1)$-CCP for certain parameters.
	CCP matrix extension	Extends a $k \times n$ CCP matrix to a $k \times (n + s(k+1))$ CCP matrix for integer s.
Codes over ring $\mathbb{Z} \mod q$	Single parity-check (SPC) code	Satisfy the $(k,k+1)$-CCP with $n = k+1$.
	Cyclic codes over the ring	Require that $q = q_1 \times q_2 \times \cdots \times q_d$ where q_i's are prime. Satisfy the (k,k)-CCP.
	Kronecker product of $z \times \alpha$ matrix satisfying the (z,z)-CCP with the identity matrix $\mathbf{I}_{t\times t}$	Satisfy the (k,k)-CCP property where $k = tz$.
	CCP matrix extension	Extends a $k \times n$ CCP matrix to a $k \times (n + s(k+1))$ CCP matrix for integer s.

Table 3.2 List of k values for Example 16. The values of n', α and z are obtained by following Algorithm 4.

k	n'	z	α	Construction	Notes
11	12	1	1	(12, 11) SPC code	$k + 1 = n'$
10	12	11	12	-	-
9	12	5	6	Claim 7	$\alpha = z + 1$ and $q \geq z$
8	12	3	4	Claim 7	$\alpha = z + 1$ and $q \geq z$
7	12	2	3	Claim 7	$\alpha = z + 1$ and $q \geq z$
6	12	7	12	Claim 4	Generator polynomial is $X^6 + X^5 + 3X^4 + 3X^3 + X^2 + 4X + 3$
5	6	1	1	(6,5) SPC code and Claim 8	Extend (6,5) SPC code to (12,5) code
4	7	5	7	Claim 4	Generator polynomial is $X^8 + X^7 + 4X^6 + 3X^5 + 2X^3 + X^2 + 4X + 4$
3	4	1	1	(4,3) SPC code and Claim 8	Extend (4,3) SPC code to (12,3) code
2	3	1	1	(3,2) SPC code and Claim 8	Extend (3,2) SPC code to (12,2) code
1	2	1	1	(2,1) SPC code and Claim 8	Extend (2,1) SPC code to (12,1) code

CHAPTER 4. ERASURE CODING FOR DISTRIBUTED MATRIX MULTIPLICATION FOR MATRICES WITH BOUNDED ENTRIES

The work of Yu et al. (2020) considers the distributed computation of the product of two large matrices $\mathbf{A}^T$ and $\mathbf{B}$, which are respectively partitioned into $p \times m$ and $p \times n$ blocks of submatrices of equal size by the master node. The key result of Yu et al. (2020) shows that the product $\mathbf{A}^T\mathbf{B}$ can be recovered as long as *any* $\tau = pmn + p - 1$ workers complete their computation.

Interestingly, similar ideas (relating matrix multiplication to polynomial interpolation) were investigated in a different context by Yagle (1995) in the mid 90's. However, the motivation for that work was fast matrix multiplication using pseudo-number theoretic transforms, rather than fault tolerance. There have been other contributions in this area Dutta et al. (2016); Lee et al. (2018, 2017); Mallick et al. (2019); Wang et al. (2018) as well, some of which predate Yu et al. (2020).

4.1 Problem Formulation and Main Contribution

4.1.1 Problem Formulation

Let $\mathbf{A}$ (size $v \times r$) and $\mathbf{B}$ (size $v \times t$) be two integer matrices[1]. We are interested in computing $\mathbf{C} \triangleq \mathbf{A}^T\mathbf{B}$ in a distributed fashion. Specifically, each worker node can store a $1/mp$ fraction of matrix $\mathbf{A}$ and a $1/np$ fraction of matrix $\mathbf{B}$. The job given to the worker node is to compute the product of the submatrices assigned to it. The master node waits for a sufficient number of the submatrix products to be communicated to it. It then determines the final result after further

[1] Floating point matrices with limited precision can be handled with appropriate scaling.

processing at its end. More precisely, matrices $\mathbf{A}$ and $\mathbf{B}$ are first block decomposed as follows:

$$\mathbf{A} = [A_{ij}], 0 \leq i < p, 0 \leq j < m, \text{ and}$$
$$\mathbf{B} = [B_{kl}], 0 \leq k < p, 0 \leq l < n,$$

where the A_{ij}'s and the B_{kl}'s are of dimension $\frac{v}{p} \times \frac{r}{m}$ and $\frac{v}{p} \times \frac{t}{n}$ respectively. The master node forms the polynomials

$$\tilde{A}(s,z) = \sum_{i,j} A_{ij} s^{\lambda_{ij}} z^{\rho_{ij}}, \text{ and}$$
$$\tilde{B}(s,z) = \sum_{k,l} B_{kl} s^{\gamma_{kl}} z^{\delta_{kl}},$$

where $\lambda_{ij}, \rho_{ij}, \gamma_{kl}$ and δ_{kl} are suitably chosen integers. Following this, the master node evaluates $\tilde{A}(s,z)$ and $\tilde{B}(s,z)$ at a fixed positive integer s and carefully chosen points $z \in \{z_1, \ldots, z_K\}$ (which can be real or complex) where K is the number of worker nodes. Note that this only requires scalar multiplication and addition operations on the part of the master node. Subsequently, it sends matrices $\tilde{A}(s,z_i)$ and $\tilde{B}(s,z_i)$ to the i-th worker node.

The i-th worker node computes the product $\tilde{A}^T(s,z_i)\tilde{B}(s,z_i)$ and sends it back to the master node. Let $1 \leq \tau \leq K$ denote the minimum number of worker nodes such that the master node can determine the required product (i.e., matrix $\mathbf{C}$) once *any* τ of the worker nodes have completed their assigned jobs. We call τ the *recovery threshold* of the scheme. In Yu et al. (2020), τ is shown to be $pmn + p - 1$.

4.1.2 Main Contribution

In this work, we demonstrate that as long as the entries in $\mathbf{A}$ and $\mathbf{B}$ are bounded by sufficiently small numbers, the recovery threshold (τ) can be significantly reduced as compared to the approach of Yu et al. (2020). Specifically, the recovery threshold in our work can be of the form $p'mn + p' - 1$ where p' is a divisor of p. Thus, we can achieve thresholds as low as mn (which is optimal), depending on our assumptions on the matrix entries. We show that the required upper bound on the matrix entries can be traded off with the corresponding threshold in a simple manner. Finally, we present experimental results that demonstrate the superiority of our method via an Amazon Web Services (AWS) implementation.

4.2 Reduced Recovery Threshold codes

4.2.1 Motivating example

Let $m = n = p = 2$ so that the following block decomposition holds

$$\mathbf{A} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \text{ and } \mathbf{B} = \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}.$$

We let

$$\tilde{A}(s,z) = A_{00} + A_{10}s^{-1} + (A_{01} + A_{11}s^{-1})z, \text{ and}$$
$$\tilde{B}(s,z) = B_{00} + B_{10}s + (B_{01} + B_{11}s)z^2.$$

The product $\tilde{A}^T(s,z)\tilde{B}(s,z)$ can be verified to be

$$\tilde{A}^T(s,z)\tilde{B}(s,z) =$$
$$s^{-1}(A_{10}^T B_{00} + A_{11}^T B_{00} z + A_{10}^T B_{01} z^2 + A_{11}^T B_{01} z^3) \tag{4.1}$$
$$+ C_{00} + C_{10}z + C_{01}z^2 + C_{11}z^3 \tag{4.2}$$
$$+ s(A_{00}^T B_{10} + A_{01}^T B_{10} z + A_{00}^T B_{11} z^2 + A_{01}^T B_{11} z^3). \tag{4.3}$$

Evidently, the product above contains the useful terms in (4.2) as coefficients of z^k for $k = 0, \ldots, 3$. The other two lines contain terms (coefficients of $s^{-1}z^k$ and sz^k, $k = 0, \ldots, 3$) that we are not interested in; we refer to these as *interference terms*. Rearranging the terms, we have

$$\tilde{A}^T(s,z)\tilde{B}(s,z) =$$
$$\underbrace{(*s^{-1} + C_{00} + *s)}_{X_{00}} + \underbrace{(*s^{-1} + C_{10} + *s)}_{X_{10}} z +$$
$$\underbrace{(*s^{-1} + C_{01} + *s)}_{X_{01}} z^2 + \underbrace{(*s^{-1} + C_{11} + *s)}_{X_{11}} z^3,$$

where $*$ denotes an interference term.

As the above polynomial is of z-degree 3, equivalently we have presented a coding strategy where we recover superposed useful and interference terms even in the presence of $K - 4$ erasures.

Now, suppose that the absolute value of each entry in $\mathbf{C}$ and of each of the interference terms is $< L$. Furthermore, assume that $s \geq 2L$. The C_{ij}'s can then be recovered by exploiting the fact that $s \geq 2L$, e.g., for non-negative matrices $\mathbf{A}$ and $\mathbf{B}$, we can simply extract the integer part of each X_{ij} and compute its remainder upon division by s. The case of general $\mathbf{A}$ and $\mathbf{B}$ is treated in Section 4.2.2.

To summarize, under our assumptions on the maximum absolute value of the matrix $\mathbf{C}$ and the interference matrix products, we can obtain a scheme with a threshold of 4. In contrast, the scheme of Yu et al. (2020) would have a threshold of 9.

Remark 7. We emphasize that the choice of polynomials $\tilde{A}(s, z)$ and $\tilde{B}(s, z)$ are quite different in our work as compared to Yu et al. (2020); this can be verified by setting $s = 1$ in the expressions. In particular, our choice of polynomials deliberately creates the controlled superposition of useful and interference terms (the choice of coefficients in Yu et al. (2020) explicitly avoids the superposition). We unentangle the superposition by using our assumptions on the matrix entries later. To our best knowledge, this unentangling idea first appeared in the work of Yagle (1995), though its motivations were different.

4.2.2 General code construction

We now present the most general form of our result. Let the block decomposed matrices $\mathbf{A}$ and $\mathbf{B}$ be of size $p \times m$ and $p \times n$ respectively. We form the polynomials $\tilde{A}(s, z)$ and $\tilde{B}(s, z)$ as follows

$$\tilde{A}(s, z) = \sum_{i=0}^{m-1} z^i \sum_{u=0}^{p-1} A_{ui} s^{-u}, \text{ and}$$
$$\tilde{B}(s, z) = \sum_{j=0}^{n-1} z^{mj} \sum_{v=0}^{p-1} B_{vj} s^{v}.$$

Under this choice of polynomials $\tilde{A}(s,z)$ and $\tilde{B}(s,z)$, we have

$$\tilde{A}^T(s,z)\tilde{B}(s,z) = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\sum_{u=0}^{p-1}\sum_{v=0}^{p-1} A_{ui}^T B_{vj} z^{mj+i} s^{v-u}. \tag{4.4}$$

To better understand the behavior of this sum, we divide it into the following cases.

- *Case 1: Useful terms.* These are the terms with coefficients of the form $A_{ui}^T B_{uj}$. They are useful since $C_{ij} = \sum_{u=0}^{p-1} A_{ui}^T B_{uj}$. It is easy to check that the term $A_{ui}^T B_{uj}$ is the coefficient of z^{mj+i}.

- *Case 2: Interference terms.* Conversely, the terms in (4.4) with coefficient $A_{ui}^T B_{vj}$, $u \neq v$ are the *interference terms* and they are the coefficients of $z^{mj+i} s^{v-u}$ (for $v \neq u$).

Based on the above discussion, we obtain

$$\begin{aligned} \tilde{A}^T(s,z)\tilde{B}(s,z) = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} z^{mj+i} \times \\ \underbrace{(*s^{-(p-1)} + \cdots + *s^{-1} + C_{ij} + *s + \cdots + *s^{p-1})}_{X_{ij}}, \end{aligned} \tag{4.5}$$

where $*$ denotes an interference term. Note that (4.5) consists of consecutive powers z^k for $k = 0, \ldots, mn-1$.

We choose distinct values z_i for worker i (real or complex). Suppose that the absolute value of each C_{ij} and of each interference term (marked with $*$) is at most $L-1$. We choose $s \geq 2L$.

4.2.3 Decoding algorithm

We now show that as long as at least mn of the worker nodes return their computations, the master node can recover the matrix $\mathbf{C}$.

Suppose the master node obtains the result $Y_i = \tilde{A}^T(s,z_i)\tilde{B}(s,z_i)$ from *any* mn workers $i_1, i_2, \ldots, i_{mn}$. Then, it can recover X_{ij}, $i = 0, \ldots, m-1, j = 0, \ldots, n-1$ by solving the following equations,

$$\begin{bmatrix} Y_{i_1} \\ Y_{i_2} \\ \vdots \\ Y_{i_{mn}} \end{bmatrix} = \begin{bmatrix} 1 & z_{i_1} & z_{i_1}^2 & \cdots & z_{i_1}^{mn-1} \\ 1 & z_{i_2} & z_{i_2}^2 & \cdots & z_{i_2}^{mn-1} \\ & & \cdots & \cdots & \\ 1 & z_{i_{mn}} & z_{i_{mn}}^2 & \cdots & z_{i_{mn}}^{mn-1} \end{bmatrix} \begin{bmatrix} X_{00} \\ X_{01} \\ \vdots \\ X_{(m-1)(n-1)} \end{bmatrix}.$$

The Vandermonde form of the above matrix guarantees the uniqueness of the solution. This is because the determinant of Vandermonde matrix can be expressed as $\prod_{1 \le a,b \le mn}(z_{i_a} - z_{i_b})$, which is non-zero since z_{i_j}, $j = 1, \cdots, mn$, are distinct.

Note that $X_{ij} = *s^{-(p-1)} + \cdots + *s^{-1} + C_{ij} + *s + \cdots + *s^{p-1}$. The master node can recover C_{ij} from X_{ij} as follows. We first round X_{ij} to the closest integer. This allows us to recover $C_{ij} + *s + \cdots + *s^{p-1}$. This is because

$$| * s^{-(p-1)} + \cdots + *s^{-1}| \le \frac{L-1}{2L-1} < 1/2.$$

Next, we determine $\hat{C}_{ij} = C_{ij} + *s + \cdots + *s^{p-1} \mod s$ (we work under the convention that the modulo output always lies between 0 and $s-1$). It is easy to see that if $\hat{C}_{ij} \le s/2$ then $C_{ij} = \hat{C}_{ij}$, otherwise C_{ij} is negative and $C_{ij} = -(s - \hat{C}_{ij})$. If s is a power of 2, the modulo operation can be performed by simple bit-shifting; this is the preferred choice.

4.2.4 Discussion of precision issues

The maximum and the minimum values (integer or floating point) that can be stored and manipulated on a computer have certain limits. Assuming $s = 2L$, it is easy to see that $|X_{ij}|$ is at most $(2L)^p/2$. Therefore, large values of L and p can potentially cause numerical issues (overflow and/or underflow). We note here that a simple but rather conservative way to estimate the value of L would be to set it equal to $v \cdot \max |\mathbf{A}| \times \max |\mathbf{B}| + 1$.

4.3 Trading off precision and threshold

The method presented in Section 4.2 achieves a threshold of mn while requiring that the LHS of (4.5) remain with the range of numeric values that can be represented on the machine. In general, the terms in (4.5) will depend on the choice of the z_i's and the values of the $|X_{ij}|$'s, e.g., choosing the z_i's to be complex roots of unity will imply that our method requires $mn \times (2L)^p/2$ to be within the range of values that can be represented.

We now present a scheme that allows us to trade off the precision requirements with the recovery threshold of the scheme, i.e., we can loosen the requirement on L and p at the cost of an increased threshold.

Assume that p' is an integer that divides p. We form the polynomials $\tilde{A}(s,z)$ and $\tilde{B}(s,z)$ as follows,

$$\tilde{A}(s,z) = \sum_{i=0}^{m-1} \sum_{j=0}^{p'-1} z^{j+p'i} \sum_{k=0}^{p/p'-1} A_{(k+\frac{p}{p'}j),i} s^k, \text{ and}$$
$$\tilde{B}(s,z) = \sum_{u=0}^{n-1} \sum_{v=0}^{p'-1} z^{mp'u+(p'-1-v)} \sum_{w=0}^{p/p'-1} B_{(w+\frac{p}{p'}v),u} s^{-w}.$$

Note that in the expressions above we use $A_{i,j}$ to represent the (i,j)-th entry of $\mathbf{A}$ (rather than A_{ij}). Next, we have

$$\tilde{A}(s,z)^T \tilde{B}(s,z) = \sum_{i=0}^{m-1} \sum_{j=0}^{p'-1} \sum_{k=0}^{p/p'-1} \sum_{u=0}^{n-1} \sum_{v=0}^{p'-1} \sum_{w=0}^{p/p'-1}$$
$$A^T_{(k+\frac{p}{p'}j),i} B_{(w+\frac{p}{p'}v),u} z^{mp'u+(p'-1-v)+j+p'i} s^{k-w}. \tag{4.6}$$

To better understand the behavior of (4.6), we again divide it into useful terms and interference terms.

- *Case 1: Useful terms.* These are the terms with coefficients of the form $A^T_{(k+\frac{p}{p'}j),i} B_{(k+\frac{p}{p'}j),u}$. The term $A^T_{(k+\frac{p}{p'}j),i} B_{(k+\frac{p}{p'}j),u}$ is the coefficient of $z^{mp'u+p'i+p'-1}$.

- *Case 2: Interference terms.* The interference terms are associated with the terms with coefficient $A^T_{(k+\frac{p}{p'}j),i} B_{(w+\frac{p}{p'}v),u}$, $k \neq w$ and/or $j \neq v$. They can be written as

$$A^T_{(k+\frac{p}{p'}j),i} B_{(w+\frac{p}{p'}v),u} z^{mp'u+(p'-1-v)+j+p'i} s^{k-w}.$$

We now verify that the interference terms and useful terms are distinct. This is evident when $k \neq w$ by examining the exponent of s. When $k = w$ but $j \neq v$ we argue as follows. Suppose that there exist some u_1, u_2, i_1, i_2 such that $mp'u_1 + p'i_1 + p' - 1 = mp'u_2 + p' + p'i_2 - v + j - 1$. Then, $mp'(u_1 - u_2) + p'(i_1 - i_2) = j - v$. This is impossible since $|j - v| < p'$.

Next, we discuss the degree of $\tilde{A}(s,z)^T\tilde{B}(s,z)$ in the variable z. In (4.6), the terms with maximal z-degree are the terms with $u = n-1, v = 0, j = p'-1$ and $i = m-1$. Thus, the maximal degree of z in the expression is $mnp' + p' - 2$. It can be verified that terms with z-degree from 0 to $mnp' + p' - 2$ will appear in (4.6) and the z-degree of the useful terms C_{iu} are $mp'u + p'i + p' - 1$, $i = 0, \cdots, m-1$, $u = 0, \cdots, n-1$.

Likewise the s-degree of $\tilde{A}(s,z)^T\tilde{B}(s,z)$ varies from $-(p-1), \ldots, 0, \ldots, (p-1)$ with the useful terms corresponding to s^0. Based on the above discussion, we obtain

$$\tilde{A}^T(s,z)\tilde{B}(s,z) = \sum_{k=0}^{mnp'+p'-2} X_k z^k, \text{where}$$
$$X_k = \begin{cases} *s^{-(\frac{p}{p'}-1)} + \cdots + *s^{-1} + C_{ij} + *s + \cdots + *s^{\frac{p}{p'}-1}, \\ \hfill \text{if } k = mp'j + p'i + p - 1 \\ *s^{-(\frac{p}{p'}-1)} + \cdots + *s^{-1} + * + *s + \cdots + *s^{\frac{p}{p'}-1}, \\ \hfill \text{otherwise.} \end{cases}$$

Evidently, the recovery threshold is $mnp' + p' - 1$, which is higher than that of the construction in Section 4.2.2. However, let $s = 2L$, the maximum value of $|X_{ij}|$ is at most $(2L)^{p/p'}/2$ which is less than the previous construction if $p' > 1$.

Example 18. Let $m = n = 2$, $p = 4$ and $p' = 2$ so that

$$\mathbf{A} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \\ A_{20} & A_{21} \\ A_{30} & A_{31} \end{bmatrix} \text{ and } \mathbf{B} = \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \\ B_{20} & B_{21} \\ B_{30} & B_{31} \end{bmatrix}.$$

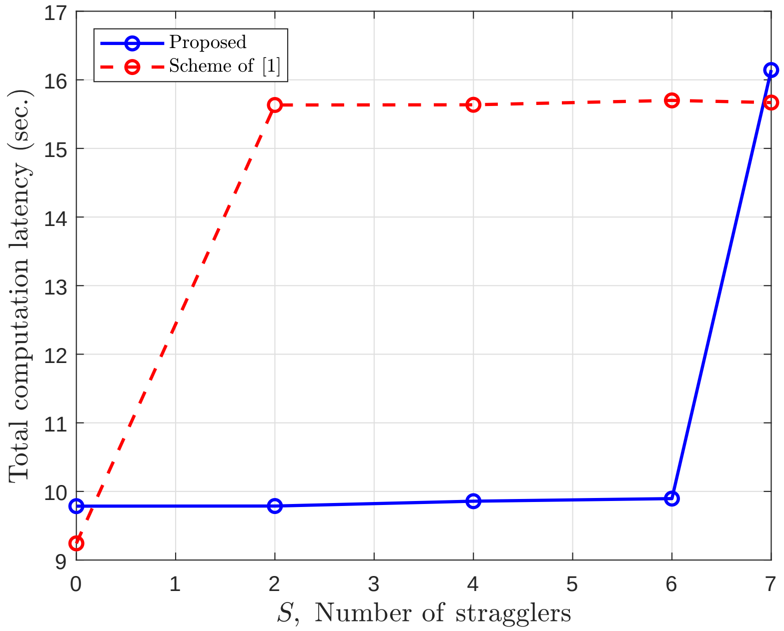

Figure 4.1 Comparison of total computation latency by simulating up to 8 stragglers

We let

$$\begin{aligned}\tilde{A}(s,z) =& A_{00} + A_{10}s^{-1} + (A_{20} + A_{30}s^{-1})z + \\ & (A_{01} + A_{11}s^{-1})z^2 + (A_{21} + A_{31}s^{-1})z^3, \text{ and} \\ \tilde{B}(s,z) =& (B_{00} + B_{10}s)z + B_{20} + B_{30}s + \\ & (B_{01} + B_{11}s)z^5 + (B_{21} + B_{31}s)z^4.\end{aligned}$$

The product of the above polynomials can be verified to contain the useful terms with coefficients z, z^3, z^5, z^7; the others are interference terms. For this scheme the corresponding $|X_{ij}|$ can at most be $2L^2$, though the recovery threshold is 9. Applying the method of Section 4.2.2 would result in the $|X_{ij}|$ values being bounded by $8L^4$ with a threshold of 4.

4.4 Experimental Results and Discussion

We ran our experiments on AWS EC2 r3.large instances. Our code is available onlinecom (2019). The input matrices **A** and **B** were randomly generated integer matrices of size 8000×8000 with elements in the set $\{0, 1, \ldots, 50\}$. These matrices were pre-generated (for the different straggler counts) and remained the same for all experiments. The master node was responsible for the 2×2 block decomposition of $\mathbf{A}$ and $\mathbf{B}$, computing $\tilde{A}(s, z_i)$ and $\tilde{B}(s, z_i)$ for $i = 1, \ldots, 10$ and sending them to the worker nodes. The evaluation points (z_i's) were chosen as 10 equally spaced reals within the interval $[-1, 1]$. The stragglers were simulated by having S randomly chosen machines perform their local computation twice.

We compared the performance of our method (*cf.* Section 4.2) with Yu et al. (2020). For fairness, we chose the same evaluation points in both methods. In fact, the choice of points in their code available online nip (2017) (which we adapted for the case when $p > 1$), provides worse results than those reported here.

Computation latency refers to the elapsed time from the point when all workers have received their inputs until enough of them finish their computations accounting for the decoding time. The decoding time for our method is slightly higher owing to the modulo s operation (*cf.* Section 4.2.3).

It can be observed in Fig. 4.1 that for our method there is no significant change in the latency for the values of $S \in \{0, 2, 4, 6\}$ and it remains around 9.83 seconds. When $S = 7$, as expected the straggler effects start impacting our system and the latency jumps to approximately 16.14 seconds. In contrast, the performance of Yu et al. (2020) deteriorates in the presence of two or more stragglers (average latency ≥ 15.65 seconds).

Real Vandermonde matrices are well-known to have bad condition numbers. The condition number is better when we consider complex Vandermonde matrices with entries from the unit circle Gautschi (1990). In our method, the $|X_{ij}|$ and $|Y_{ij}|$ values can be quite large. This introduces small errors in the decoding process. Let $\hat{\mathbf{C}}$ be the decoded matrix and $\mathbf{C} \triangleq \mathbf{A}^T\mathbf{B}$ be the actual product. Our error metric is $e = \frac{||\mathbf{C} - \hat{\mathbf{C}}||_F}{||\mathbf{C}||_F}$ (subscript F refers to the Frobenius norm). The results in Fig. 1, had an error e of at most 10^{-7}. We studied the effect of increasing the average value of the entries in $\mathbf{A}$ and $\mathbf{B}$ in Table 1. The error is consistently low up to a bound of $L = 1000$, following which the calculation is useless owing to numerical overflow issues. We point out that in our experiments

Table 4.1 Effect of bound (L) on the decoding error

Bound(L)	s	Error
100	2^{28}	$6.31 \cdot 10^{-7}$
200	2^{30}	$8.87 \cdot 10^{-7}$
500	2^{32}	$6.40 \cdot 10^{-6}$
1000	2^{34}	$9.52 \cdot 10^{-6}$
2000	2^{36}	1

the error e was identically zero if the z_i's were chosen from the unit circle. However, this requires complex multiplication, which increases the computation time.

CHAPTER 5. NUMERICALLY STABLE CODED MATRIX COMPUTATIONS VIA CIRCULANT AND ROTATION MATRIX EMBEDDINGS

Polynomial based methods have been used in several works for mitigating the effect of stragglers in distributed matrix computations. However, they suffer from serious numerical issues owing to the condition number of the corresponding real Vandermonde-structured recovery matrices. In this chapter, we present a novel approach that leverages the properties of circulant permutation matrices and rotation matrices for coded matrix computation. In addition to having an optimal recovery threshold, our scheme has condition numbers that are orders of magnitude lower than prior work.

5.1 Problem Setting, Related Work and Main contributions

5.1.1 Problem Setting

Consider a scenario where the master node has a large $t \times r$ matrix $\mathbf{A} \in \mathbb{R}^{t\times r}$ and either a $t \times 1$ vector $\mathbf{x} \in \mathbb{R}^{t\times 1}$ or a $t \times w$ matrix $\mathbf{B} \in \mathbb{R}^{t\times w}$. The master node wishes to compute $\mathbf{A}^T\mathbf{x}$ or $\mathbf{A}^T\mathbf{B}$ in a distributed manner over n worker nodes in the matrix-vector and matrix-matrix setting respectively. Towards this end, the master node partitions $\mathbf{A}$ (respectively $\mathbf{B}$) into Δ_A (respectively Δ_B) block columns. Each worker node is assigned $\delta_A \leq \Delta_A$ and $\delta_B \leq \Delta_B$ linearly encoded block columns of $\mathbf{A}_0, \dots, \mathbf{A}_{\Delta_A - 1}$ and $\mathbf{B}_0, \dots, \mathbf{B}_{\Delta_B - 1}$, so that $\delta_A/\Delta_A \leq \gamma_A$ and $\delta_B/\Delta_B \leq \gamma_B$, where γ_A and γ_B represent storage capacity constraints for $\mathbf{A}$ and $\mathbf{B}$ respectively.

In the matrix-vector case, the i-th worker is assigned encoded block-columns of $\mathbf{A}$ and the vector $\mathbf{x}$ and computes their inner product. In the matrix-matrix case it computes all pairwise products of block-columns assigned to it. We say that a given scheme has *computation threshold* τ if the master node can decode the intended result as long as *any* τ out of n worker nodes complete their jobs. In this case we say that the scheme is resilient to $s = n - \tau$ stragglers. We say that this threshold is *optimal* if the value of τ is the smallest possible for the given storage capacity constraints.

The overall goal is to (i) design schemes that are resilient to s stragglers (s is a design parameter), while ensuring that the (ii) desired result can be decoded in a efficient manner, and (iii) the decoded result is numerically robust even in the presence of round-off errors and other sources of noise.

5.1.2 Related Work

A significant amount of prior work Yu et al. (2017, 2020); Dutta et al. (2016, 2019) has demonstrated interesting and elegant approaches based on embedding the distributed matrix computation into the structure of polynomials. Specifically, the encoding at the master node can be viewed as evaluating certain polynomials at distinct real values. Each worker node gets a particular evaluation. Once, at least τ workers finish their tasks, the master node can decode the intended result by performing polynomial interpolation. The work of Yu et al. (2017) demonstrates that when $\delta_A = \delta_B = 1$ the optimal threshold is $\Delta_A \Delta_B$ and that polynomial based approaches (henceforth referred to as polynomial codes) achieve this threshold. Prior work has also considered other ways in which matrices $\mathbf{A}$ and $\mathbf{B}$ can be partitioned. For instance, they can be partitioned both along rows and columns. The work of Yu et al. (2020); Dutta et al. (2019) has obtained threshold results in those cases as well. The so called Entangled Polynomial and Mat-Dot codes Yu et al. (2020); Dutta et al. (2019), also use polynomial encodings. The key point is that in all these approaches, polynomial interpolation is required when decoding the required result.

A major part of our work in this paper revolves around understanding the numerical stability of various distributed matrix computations. This is closely related to the condition number of matrices. Let $||\mathbf{M}||$ denote the maximum singular value of a matrix $\mathbf{M}$ of dimension $l \times l$.

Definition 7. *Condition number.* The condition number of a $l \times l$ matrix $\mathbf{M}$ is defined as $\kappa(\mathbf{M}) = ||\mathbf{M}||||\mathbf{M}^{-1}||$. It is infinite if the minimum singular value of $\mathbf{M}$ is zero.

Consider the system of equations $\mathbf{My} = \mathbf{z}$, where $\mathbf{z}$ is known and $\mathbf{y}$ is to be determined. If $\kappa(\mathbf{M}) \approx 10^b$, then the decoded result loses approximately b digits of precision Higham (2002). In particular, matrices that are ill-conditioned lead to significant numerical problems when solving linear equations.

Polynomial interpolation corresponds to solving a real *Vandermonde* system of equations at the master node. In the work of Yu et al. (2017), this would require solving a $\Delta_A\Delta_B \times \Delta_A\Delta_B$ Vandermonde system. Unfortunately, it can be shown that the condition number of these matrices grows exponentially in $\Delta_A\Delta_B$ Pan (2016). This is a significant drawback and even for systems with around $\Delta_A\Delta_B \approx 30$, the condition number is so high that the decoded results are essentially useless.

In Section VII of Yu et al. (2020), it is remarked that when operating over infinite fields such as the reals, one can embed the computation into finite fields to avoid numerical errors. For instance, they advocate encoding and decoding over a finite field of order p. However, this method would require "quantizing" real matrices $\mathbf{A}$ and $\mathbf{B}$ so that the entries are integers. We demonstrate that the performance of this method can be catastrophically bad.

For this method to work, the maximum possible absolute value of each entry of the quantized matrices, α should be such that $\alpha^2 t < p$, since each entry corresponds to the inner product of columns of $\mathbf{A}$ and columns of $\mathbf{B}$. This "dynamic range constraint (DRC)" means that the error in the computation depends strongly on the actual matrix entries and the value of t is quite limited. If the DRC is violated, the error in the underlying computation can be catastrophic. Even if the DRC is not violated, the dependence of the error on the entries can make it very bad. We discuss the complete details in Section 5.6.

The main goal of our work is to consider alternate embeddings of distributed matrix computations that are based on rotation and circulant permutation matrices. We demonstrate that these are significantly better behaved from a numerical precision perspective.

The issue of numerical stability in the coded computation context has been considered in a few recent works Tang et al. (2019); Ramamoorthy et al. (2019); Das et al. (2018); Das and Ramamoorthy (2019); Das et al. (2019); Fahim and Cadambe (2019); Subramaniam et al. (2019). The work of Ramamoorthy et al. (2019); Das and Ramamoorthy (2019) presented strategies for distributed matrix-vector multiplication and demonstrated some schemes that empirically have better numerical performance than polynomial based schemes for some values of n and s. However, both these approaches work only for the matrix-vector problem. Preprint Das et al. (2019) presents a random convolutional coding approach that applies for both the matrix-vector and the matrix-matrix multiplications problems. Their work demonstrates a computable upper bound on the worst case condition number of the decoding matrices by drawing on connections with the asymptotic analysis of large Toeplitz matrices. The recent preprint Subramaniam et al. (2019) presents constructions that are based on random linear coding ideas where the encoding coefficients are chosen at random from a continuous distribution. These exhibit better condition number properties.

The work most closely related to the our work is Fahim and Cadambe (2019) considers an alternative approach for polynomial based schemes by working within the basis of orthogonal polynomials. They demonstrate an upper bound on the worst case condition number of the decoding matrices which grows as $O(n^{2s})$ where s is the number of stragglers that the scheme is resilient to. They also demonstrate experimentally that their performance is significantly better than the polynomial code approach. In contrast we demonstrate an upper bound that is $\approx O(n^{s+6})$. Furthermore, in Section 5.6 we show that in practice our worst case condition numbers are far better than Fahim and Cadambe (2019).

5.1.3 Main contributions

The work of Pan (2016) shows that unless all (or almost all) the parameters of the Vandermonde matrix lie on the unit circle, its condition number is badly behaved. However, most of these parameters are complex-valued (except ±1), whereas our matrices **A** and **B** are real-valued. Using complex evaluation points in the polynomial code scheme, will increase the cost of computations approximately four times for matrix-matrix multiplication and around two times for matrix-vector multiplication. This is an unacceptable hit in computation time.

- Our main finding in this paper is that we can work with real-valued matrix embeddings that (i) continue to have the *optimal* threshold of polynomial based approaches, and (ii) enjoy the low condition number of complex Vandermonde matrices with all parameters on the unit circle. In particular, we demonstrate that rotation matrices and circulant permutation matrices of appropriate sizes can be used within the framework of polynomial codes. At the top level, instead of evaluating polynomials at real values, our approach evaluates the polynomials at matrices.

- Using these embeddings we show that the worst case condition number over all $\binom{n}{n-s}$ possible recovery matrices is upper bounded by $\approx O(n^{s+6})$. Furthermore, our experimental results indicate that the actual values are significantly smaller, i.e., the analysis yields pessimistic upper bounds.

5.2 Distributed Matrix Computation Schemes

Our schemes in this work will be defined by the encoding matrices used by the master node, which are such that the master node only needs to perform scalar multiplications and additions. The computationally intensive tasks, i.e., matrix operations are performed by the worker nodes. We begin by defining certain classes of matrices, discuss their relevant properties and present an example that outlines the basic idea of our work. We let $\mathrm{i} = \sqrt{-1}$.

Definition 8. *Rotation matrix.* The 2×2 matrix $\mathbf{R}_\theta$ below is called a rotation matrix.

$$\mathbf{R}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = \mathbf{Q}\Lambda\mathbf{Q}^*, \text{ where} \tag{5.1}$$

$$\mathbf{Q} = \frac{1}{\sqrt{2}} \begin{bmatrix} \mathrm{i} & -\mathrm{i} \\ 1 & 1 \end{bmatrix}, \text{ and } \Lambda = \begin{bmatrix} e^{\mathrm{i}\theta} & 0 \\ 0 & e^{-\mathrm{i}\theta} \end{bmatrix}. \tag{5.2}$$

Definition 9. *Circulant Permutation Matrix.* Let $\mathbf{e}$ be a row vector of length m with $\mathbf{e} = [0\ 1\ 0\ \ldots\ 0]$. Let $\mathbf{P}$ be a $m \times m$ matrix with $\mathbf{e}$ as its first row. The remaining rows are obtained by cyclicly shifting the first row with the shift index equal to the row index. Then $\mathbf{P}^i, i \in [m]$ are said to be circulant permutation matrices. Let $\mathbf{W}$ denote the m-point Discrete Fourier Transform (DFT) matrix, i.e., $\mathbf{W}(i,j) = \frac{1}{\sqrt{m}}\omega_m^{ij}$ for $i \in [m], j \in [m]$ where $\omega_m = e^{-\mathrm{i}\frac{2\pi}{\mathrm{m}}}$ denotes the m-th root of unity. Then, it can be shown Gray (2006) that $\mathbf{P} = \mathbf{W}\text{diag}(1, \omega_m, \omega_m^2, \ldots, \omega_m^{(m-1)})\mathbf{W}^*$, where the superscript $*$ denotes the complex conjugate operator.

Remark 8. Rotation matrices and circulant permutation matrices (see Appendix B for an example) have the useful property that they are "real" matrices with complex eigenvalues lie on the unit circle. We use this property extensively in the sequel.

Illustrative Example: Consider the matrix-vector case where $\Delta_A = 3$ and $\delta_A = 1$. In the polynomial approach, the master node forms $\mathbf{A}(z) = \mathbf{A}_0 + \mathbf{A}_1 z + \mathbf{A}_2 z^2$ and evaluates it at distinct real values $z_1, \ldots, z_n$. The i-th evaluation is sent to the i-th worker node which computes $\mathbf{A}^T(z_i)\mathbf{x}$. From polynomial interpolation, it follows that as long as the master node receives results from any three workers, it can decode $\mathbf{A}^T\mathbf{x}$. However, when Δ_A is large, the interpolation is numerically unstable.

The basic idea of our approach is as follows. We further split each $\mathbf{A}_i$ into two equal sized block columns. Thus we now have six block-columns, indexed as $\mathbf{A}_0, \ldots \mathbf{A}_5$. Consider the 6×2 matrix defined below; its columns are specified by $\mathbf{g}_0$ and $\mathbf{g}_1$.

$$[\mathbf{g}_0 \ \mathbf{g}_1] = \begin{bmatrix} \mathbf{I} \\ \mathbf{R}_\theta^i \\ \mathbf{R}_\theta^{2i} \end{bmatrix}$$

The master node forms "two" encoded matrices for the i-th worker: $\sum_{j=0}^{5} \mathbf{A}_j \mathbf{g}_0(j)$ and $\sum_{j=0}^{5} \mathbf{A}_j \mathbf{g}_1(j)$ (where $\mathbf{g}_i(l)$ denotes the l-th component of the vector $\mathbf{g}_i$). Thus, the storage capacity constraint fraction γ_A is still $1/3$. Worker node i computes the inner product of these two encoded matrices with $\mathbf{x}$ and sends the result to the master node. It turns out that in this case when any three workers i_0, i_1, and i_2 complete their tasks, the decodability and numerical stability of recovering $\mathbf{A}^T\mathbf{x}$ depends on the condition number of the following matrix.

$$\begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} \\ \mathbf{R}_\theta^{i_0} & \mathbf{R}_\theta^{i_1} & \mathbf{R}_\theta^{i_2} \\ \mathbf{R}_\theta^{2i_0} & \mathbf{R}_\theta^{2i_1} & \mathbf{R}_\theta^{2i_2} \end{bmatrix}$$

Using the eigendecomposition of $\mathbf{R}_\theta$ (*cf.* 5.6) the above block matrix can expressed as

$$\begin{bmatrix} \mathbf{Q} & 0 & 0 \\ 0 & \mathbf{Q} & 0 \\ 0 & 0 & \mathbf{Q} \end{bmatrix} \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} \\ \Lambda^{i_0} & \Lambda^{i_1} & \Lambda^{i_2} \\ \Lambda^{2i_0} & \Lambda^{2i_1} & \Lambda^{2i_2} \end{bmatrix}}_{\mathbf{\Sigma}} \begin{bmatrix} \mathbf{Q}^* & 0 & 0 \\ 0 & \mathbf{Q}^* & 0 \\ 0 & 0 & \mathbf{Q}^* \end{bmatrix},$$

As the pre- and post-multiplying matrices are unitary, the condition number of the above matrix only depends on the properties of the middle matrix, denoted by $\mathbf{\Sigma}$. In what follows we show that upon appropriate column and row permutations, $\mathbf{\Sigma}$ can be shown equivalent to a block diagonal matrix where each of the blocks is a Vandermonde matrix with parameters on the unit circle. Thus, the matrix is invertible. Furthermore, even though we use real computation, the numerical stability of our scheme depends on Vandermonde matrices with parameters on the unit circle. The upcoming Theorem 3 shows that the condition number of such matrices is much better behaved.

In the sequel we show that this argument can be significantly generalized and adapted for the case of circulant permutation embeddings. The matrix-matrix case requires the development of more ideas that we also present.

Notation: Let $[m]$ denote the set $\{0, \ldots, m-1\}$. For a matrix $\mathbf{M}$, $\mathbf{M}(i,j)$ denotes its (i,j)-th entry, whereas $\mathbf{M}_{i,j}$ denotes the (i,j)-th block sub-matrix of $\mathbf{M}$. We use MATLAB inspired notation at certain places. For instance, $\text{diag}(a_1, a_2, \ldots, a_m)$ denotes a $m \times m$ diagonal matrix with a_i's on the diagonal and $\mathbf{M}(:,j)$ denotes the j-th column of matrix $\mathbf{M}$. The notation $\mathbf{M}_1 \otimes \mathbf{M}_2$ denotes the Kronecker product of $\mathbf{M}_1$ and $\mathbf{M}_2$.

Encoding schemes: In this work our general strategy will be to first partition the matrices $\mathbf{A}$ and $\mathbf{B}$ into $\Delta_A = k_A\ell$ and $\Delta_B = k_B\ell$ block-columns respectively. However, we use two indices to refer to their respective constituent block columns as this simplifies our later presentation. To avoid confusion, we use the subscript $\langle i,j\rangle$ to refer to the corresponding (i,j)-th block-columns. In particular $\mathbf{A}_{\langle i,j\rangle}, i \in [k_A], j \in [\ell]$ and $\mathbf{B}_{\langle i,j\rangle}, i \in [k_B], j \in [\ell]$ refer to the (i,j)-th block column of $\mathbf{A}$ and $\mathbf{B}$ respectively, such that

$$\begin{aligned} \mathbf{A} &= [\mathbf{A}_{\langle 0,0\rangle} \;\ldots\; \mathbf{A}_{\langle 0,\ell-1\rangle} \mid \ldots \mid \mathbf{A}_{\langle k_A-1,0\rangle} \;\ldots\; \mathbf{A}_{\langle k_A-1,\ell-1\rangle}], \\ \mathbf{B} &= [\mathbf{B}_{\langle 0,0\rangle} \;\ldots\; \mathbf{B}_{\langle 0,\ell-1\rangle} \mid \ldots \mid \mathbf{A}_{\langle k_B-1,0\rangle} \;\ldots\; \mathbf{A}_{\langle k_B-1,\ell-1\rangle}]. \end{aligned} \tag{5.3}$$

The encoding matrix for $\mathbf{A}$ will be specified by a $k_A\ell \times n\ell$ "generator" matrix $\mathbf{G}$ such that

$$\hat{\mathbf{A}}_{\langle i,j\rangle} = \sum_{\alpha\in[k_A],\beta\in[\ell]} \mathbf{G}(\alpha\ell+\beta, i\ell+j)\mathbf{A}_{\langle\alpha,\beta\rangle} \tag{5.4}$$

for $i \in [n], j \in [\ell]$. A similar rule will apply for $\mathbf{B}$ and result in encoded matrices $\hat{\mathbf{B}}_{\langle i,j\rangle}$. Thus, in the matrix-vector case worker node i stores $\hat{\mathbf{A}}_{\langle i,j\rangle}$ for $j \in [\ell]$ and $\mathbf{x}$, whereas in the matrix-matrix case it stores $\hat{\mathbf{A}}_{\langle i,j\rangle}$ and $\hat{\mathbf{B}}_{\langle i,j\rangle}$, for $j \in [\ell]$. Thus worker i stores $\gamma_A = \ell/\Delta_A = 1/k_A$ and $\gamma_B = \ell/\Delta_B = 1/k_B$ fractions of matrices $\mathbf{A}$ and $\mathbf{B}$ respectively. In the matrix-vector case, worker node i computes $\hat{\mathbf{A}}^T_{\langle i,j\rangle}\mathbf{x}$ for $j \in [\ell]$ and transmits them to the master node. In the matrix-matrix case, it computes all ℓ^2 pairwise products $\hat{\mathbf{A}}^T_{\langle i,l_1\rangle}\hat{\mathbf{B}}_{\langle i,l_2\rangle}$ for $l_1 \in [\ell], l_2 \in [\ell]$.

Decoding Scheme: With the above encoding, the decoding process corresponds to solving linear equations. We discuss the matrix-vector case here; the matrix-matrix case is quite similar. In the matrix-vector case, the master node receives $\hat{\mathbf{A}}^T_{\langle i,j\rangle}\mathbf{x}$ of length r/Δ_A for $j \in [\ell]$ from a certain number of worker nodes and wants to decode $\mathbf{A}^T\mathbf{x}$ of length r. Based on our encoding scheme, this can be done by solving a $\Delta_A \times \Delta_A$ linear system of equations r/Δ_A times. The structure of this linear system is inherited from the encoding matrix $\mathbf{G}$.

Definition 10. *Vandermonde Matrix.* A $m\times m$ Vandermonde matrix $\mathbf{V}$ with parameters $z_1, z_2, \ldots, z_m \in \mathbb{C}$ is such that $\mathbf{V}(i,j) = z_j^i, i \in [m], j \in [m]$. If the z_i's are distinct, then $\mathbf{V}$ is nonsingular Horn and Johnson (1991). In this work, we will also assume that the z_i's are non-zero.

Condition Number of Vandermonde Matrices: Let $\mathbf{V}$ be a $m \times m$ Vandermonde matrix with parameters $s_0, s_1, \ldots s_{m-1}$. The following facts about $\kappa(\mathbf{V})$ follow from prior work Pan (2016).

- *Real Vandermonde matrices.* If $s_i \in \mathbb{R}, i \in [m]$, i.e., if $\mathbf{V}$ is a real Vandermonde matrix, then it is known that its condition number is exponential in m.

- *Complex Vandermonde matrices with parameters "not" on the unit circle.* Suppose that the s_i's are complex and let $s_+ = \max_{i=0}^{m-1} |s_i|$. If $s_+ > 1$ then $\kappa(\mathbf{V})$ is exponential in m. Furthermore, if $1/|s_i| \geq \nu > 1$ for at least $\beta \leq m$ of the m parameters, then $\kappa(\mathbf{V})$ is exponential in β.

Based on the above facts, the only scenario where the condition number is somewhat well-behaved is if most or all of the parameters of $\mathbf{V}$ are complex and lie on the unit-circle. In the Appendix B, we show the following result which is one of our key technical contributions.

Theorem 3. Consider a $m \times m$ Vandermonde matrix $\mathbf{V}$ where $m < q$ (where q is odd) with distinct parameters $\{s_0, s_1, \ldots, s_{m-1}\} \subset \{1, \omega_q, \omega_q^2, \ldots, \omega_q^{q-1}\}$. Then,

$$\kappa(\mathbf{V}) \leq O(q^{q-m+6}).$$

Remark 9. If $q - m$ is a constant, then $\kappa(\mathbf{V})$ grows only polynomially in q. In the subsequent discussion, we will leverage Theorem 3 extensively.

5.3 Distributed Matrix-Vector Multiplication

5.3.1 Rotation Matrix Embedding

Let q be an odd number such that $q \geq n$, $\theta = 2\pi/q$ and $\ell = 2$ (*cf.* block column decomposition in (5.3)). We choose the generator matrix such that its (i,j)-th block submatrix for $i \in [k_A], j \in [n]$ is given by

$$\mathbf{G}^{rot}_{i,j} = \mathbf{R}^{ji}_\theta \tag{5.5}$$

Theorem 4. The threshold for the rotation matrix based scheme specified above is k_A. Furthermore, the worst case condition number of the recovery matrices is upper bounded by $O(q^{q-k_A+6})$.

Proof. Suppose that workers indexed by $i_0, \ldots, i_{k_A-1}$ complete their tasks. We extract the corresponding block columns of $\mathbf{G}^{rot}$ to obtain

$$\tilde{\mathbf{G}}^{rot} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} \\ \mathbf{R}^{i_0}_\theta & \mathbf{R}^{i_1}_\theta & \cdots & \mathbf{R}^{i_{k_A-1}}_\theta \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{R}^{i_0(k_A-1)}_\theta & \mathbf{R}^{i_1(k_A-1)}_\theta & \cdots & \mathbf{R}^{i_{k_A-1}(k_A-1)}_\theta \end{bmatrix}.$$

We note here that the decoder attempts to recover each entry of $\mathbf{A}^T_{\langle i,j\rangle}\mathbf{x}$ from the results sent by the worker nodes. Thus, we can equivalently analyze the decoding by considering the system of equations as

$$\mathbf{m}\tilde{\mathbf{G}}^{rot} = \mathbf{c},$$

where $\mathbf{m}, \mathbf{c} \in \mathbb{R}^{1\times k_A \ell}$ are row-vectors such that

$$\begin{aligned}\mathbf{m} &= [\mathbf{m}_0, \cdots, \mathbf{m}_{k_A-1}] \\ &= [\mathbf{m}_{\langle 0,0\rangle}, \cdots, \mathbf{m}_{\langle 0,\ell-1\rangle}, \cdots, \mathbf{m}_{\langle k_A-1,0\rangle}, \cdots, \mathbf{m}_{\langle k_A-1,\ell-1\rangle}],\end{aligned}$$

and

$$\begin{aligned}\mathbf{c} &= [\mathbf{c}_{i_0}, \cdots, \mathbf{c}_{i_{k_A-1}}] \\ &= [\mathbf{c}_{\langle i_0,0\rangle}, \cdots, \mathbf{c}_{\langle i_0,\ell-1\rangle}, \cdots, \mathbf{c}_{\langle i_{k_A-1},0\rangle}, \cdots, \mathbf{c}_{\langle i_{k_A-1},\ell-1\rangle}].\end{aligned}$$

In the expression above, terms of the form $\mathbf{m}_{\langle i,j\rangle}$ and $\mathbf{c}_{\langle i,j\rangle}$ are scalars. We need to analyze $\kappa(\tilde{\mathbf{G}}^{rot})$. Towards this end, using the eigenvalue decomposition of $\mathbf{R}_\theta$, we have

$$\tilde{\mathbf{G}}^{rot} = \begin{bmatrix}\mathbf{Q} & & \\ & \ddots & \\ & & \mathbf{Q}\end{bmatrix} \tilde{\mathbf{\Lambda}} \begin{bmatrix}\mathbf{Q}^* & & \\ & \ddots & \\ & & \mathbf{Q}^*\end{bmatrix}, \text{ where} \tag{5.6}$$

$$\tilde{\mathbf{\Lambda}} = \begin{bmatrix}\mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} \\ \Lambda^{i_0} & \Lambda^{i_1} & \cdots & \Lambda^{i_{k_A-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \Lambda^{i_0(k_A-1)} & \Lambda^{i_1(k_A-1)} & \cdots & \Lambda^{i_{k_A-1}(k_A-1)}\end{bmatrix}$$

and Λ is specified in (5.2). Note that the pre- and post-multiplying matrices in (5.6) above are both unitary. Therefore $\kappa(\tilde{\mathbf{G}}^{rot})$ is the same as $\kappa(\tilde{\mathbf{\Lambda}})$ Horn and Johnson (1991).

Using Claim 14 in the Appendix B, we can permute $\tilde{\mathbf{\Lambda}}$ to put it in block-diagonal form so that

$$\tilde{\mathbf{\Lambda}}_d = \begin{bmatrix}\tilde{\mathbf{\Lambda}}_d[0] & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{\Lambda}}_d[1]\end{bmatrix},$$

Algorithm 5: Decoding Algorithm for Circulant Permutation Scheme

```
Input: $\mathbf{G}_{\mathcal{I}}^{circ}$ where $|\mathcal{I}| = k_A$ (block-columns of $G$ corresponding to block-columns in $\mathcal{I}$). $\mathbf{c}$
    corresponding to observed values in one system of equations.
Output: $\mathbf{m}$ which is the solution to $\mathbf{m}\mathbf{G}_{\mathcal{I}}^{circ} = \mathbf{c}$.
1. procedure: Block Fourier Transform and permute $\mathbf{c}$.
for $j = 0$ to $k_A - 1$ do
    Apply FFT to $\mathbf{c}_{i_j} = [\mathbf{c}_{\langle i_j,0\rangle}, \cdots, \mathbf{c}_{\langle i_j,\tilde{q}-1\rangle}]$ to obtain $\mathbf{c}_{i_j}^{\mathcal{F}} = [\mathbf{c}_{\langle i_j,0\rangle}^{\mathcal{F}}, \cdots, \mathbf{c}_{\langle i_j,\tilde{q}-1\rangle}^{\mathcal{F}}]$.
end
Permute $\mathbf{c}^{\mathcal{F}} = [\mathbf{c}_{i_0}^{\mathcal{F}}, \cdots, \mathbf{c}_{i_{k_A-1}}^{\mathcal{F}}]$ by $\pi$ to obtain $\mathbf{c}^{\mathcal{F},\pi} = [\mathbf{c}_0^{\mathcal{F},\pi}, \cdots, \mathbf{c}_{\tilde{q}-1}^{\mathcal{F},\pi}]$ where
    $\mathbf{c}_j^{\mathcal{F},\pi} = [\mathbf{c}_{\langle i_0,j\rangle}^{\mathcal{F}}, \mathbf{c}_{\langle i_1,j\rangle}^{\mathcal{F}}, \cdots, \mathbf{c}_{\langle i_{k_A-1},j\rangle}^{\mathcal{F}}]$, for $j = 0, \dots, \tilde{q} - 1$.
2. procedure: Decode $\mathbf{m}^{\mathcal{F},\pi}$ from $\mathbf{c}^{\mathcal{F},\pi}$.
For $i \in \{1, \dots, \tilde{q} - 1\}$, decode $\mathbf{m}_i^{\mathcal{F},\pi}$ from $\mathbf{c}_i^{\mathcal{F},\pi}$ by polynomial interpolation or matrix
    inversion of $\tilde{\mathbf{G}}_d^{\mathcal{F}}[i]$ (see (B.2) in Appendix B). Set $\mathbf{m}_0^{\mathcal{F},\pi} = [0, \cdots, 0]$.
3. procedure: Inverse permute and Block Inverse Fourier Transform $\mathbf{m}^{\mathcal{F},\pi}$.
Permute $\mathbf{m}^{\mathcal{F},\pi}$ by $\pi^{-1}$ to obtain $\mathbf{m}^{\mathcal{F}} = [\mathbf{m}_0^{\mathcal{F}}, \cdots, \mathbf{m}_{k_A-1}^{\mathcal{F}}]$. Apply inverse FFT to each $\mathbf{m}_i^{\mathcal{F}}$
    in $\mathbf{m}^{\mathcal{F}}$ to obtain $\mathbf{m} = [\mathbf{m}_0, \cdots, \mathbf{m}_{k_A-1}]$.
```

where $\tilde{\mathbf{\Lambda}}_d[0]$ and $\tilde{\mathbf{\Lambda}}_d[1]$ are Vandermonde matrices with parameter sets $\{e^{\mathrm{i}\theta i_0}, \dots, e^{\mathrm{i}\theta i_{k_A-1}}\}$ and $\{e^{-\mathrm{i}\theta i_0}, \dots, e^{-\mathrm{i}\theta i_{k_A-1}}\}$ respectively. Note that these parameters are distinct points on the unit circle. Thus, $\tilde{\mathbf{\Lambda}}_d[0]$ and $\tilde{\mathbf{\Lambda}}_d[1]$ are both invertible which implies that $\tilde{\Lambda}$ is invertible. This allows us to conclude that the threshold of the scheme is k_A. The upper bound on the condition number follows from Theorem 3. □

We note here that the decoding process involves inverting a $\Delta_A \times \Delta_A$ matrix once and using the inverse to solve r/Δ systems of equations. Thus, the overall decoding complexity is $O(\Delta_A^3 + r\Delta_A)$ where typically, $r \gg \Delta_A^2$.

5.3.2 Circulant Permutation Embedding

Let $\tilde{q}$ be a *prime* number which is greater than or equal to n. We set $\ell = \tilde{q} - 1$, so that $\mathbf{A}$ is sub-divided into $k_A(\tilde{q} - 1)$ block-columns as in (5.3). In this embedding we have an additional step. Specifically, the master node generates the following "precoded" matrices.

$$\mathbf{A}_{\langle i,\tilde{q}-1\rangle} = -\sum_{j=0}^{\tilde{q}-2} \mathbf{A}_{\langle i,j\rangle}, i \in [k_A]. \tag{5.7}$$

In the subsequent discussion, we work with the set of block-columns $\mathbf{A}_{\langle i,j \rangle}$ for $i \in [k_A], j \in [\tilde{q}]$. The coded submatrices $\hat{\mathbf{A}}_{\langle i,j \rangle}$ for $i \in [n], j \in [\tilde{q}]$ are generated by means of a $k_A\tilde{q} \times n\tilde{q}$ matrix $\mathbf{G}^{circ}$ as follows.

$$\hat{\mathbf{A}}_{\langle i,j \rangle} = \sum_{\alpha \in [k_A], \beta \in [\tilde{q}]} \mathbf{G}^{circ}(\alpha\tilde{q} + \beta, i\tilde{q} + j)\mathbf{A}_{\langle \alpha,\beta \rangle}, \tag{5.8}$$

where the (i, j)-th block of $\mathbf{G}^{circ}$ can be expressed as

$$\mathbf{G}^{circ}_{i,j} = \mathbf{P}^{ji}, \text{ for } i \in [k_A], j \in [n]. \tag{5.9}$$

The matrix $\mathbf{P}$ denotes the $\tilde{q} \times \tilde{q}$ circulant permutation matrix introduced in Definition 9. For this scheme the storage fraction $\gamma_A = \tilde{q}/(k_A(\tilde{q} - 1))$, i.e., it is slightly higher than $1/k_A$.

Remark 10. The $\hat{\mathbf{A}}_{\langle i,j \rangle}$'s can simply be generated by additions since $\mathbf{G}^{circ}$ is a binary matrix.

Theorem 5. The threshold for the circulant permutation based scheme specified above is k_A. Furthermore, the worst case condition number of the recovery matrices is upper bounded by $O(\tilde{q}^{\tilde{q}-k_A+6})$ and the scheme can be decoded by using Algorithm 5.

The proof appears in the Appendix B. It is conceptually similar to the proof of Theorem 4 and relies critically on the fact that all eigenvalues of $\mathbf{P}$ lie on the unit circle and that $\mathbf{P}$ can be diagonalized by the DFT matrix $\mathbf{W}$. It suggests an efficient decoding algorithm where the fast Fourier Transform (FFT) plays a key role (see Algorithm 5 and Claim 12).

Claim 12. The decoding complexity of recovering $\mathbf{A}^T\mathbf{x}$ is $O(r(\log\tilde{q} + \log^2 k_A))$.

Remark 11. Both circulant permutation matrices and rotation matrices allow us to achieve a specified threshold for distributed matrix vector multiplication. The required storage fraction γ_A is slightly higher for the circulant permutation case and it requires q to be prime. However, it allows for an efficient FFT based decoding algorithm. On the other hand, the rotation matrix case requires a smaller Δ_A, but the decoding requires solving the corresponding system of equations the complexity of which can be cubic in Δ_A. We note that when the matrix sizes are large, the decoding time will be negligible as compared to the worker node computation time; we discuss this in Section 5.6. In Section 5.6, we show results that demonstrate that the normalized mean-square error when circulant permutation matrices are used is lower than the rotation matrix case.

The circulant matrix embedding idea can be also applied to the fast encoder of Quasi-cyclic (QC) LDPC code in Huang et al. (2014); Tang et al. (2013); Zhang et al. (2014).

5.4 Distributed Matrix-Matrix Multiplication

The matrix-matrix case requires the introduction of newer ideas within this overall framework. In this case, a given worker obtains encoded block-columns of both $\mathbf{A}$ and $\mathbf{B}$ and representing the underlying computations is somewhat more involved. Once again we let $\theta = 2\pi/q$, where $q \geq n$ (n is the number of worker nodes) is an odd integer and set $\ell = 2$. Furthermore, let $k_A k_B < n$. The (i,j)-th blocks of the encoding matrices are given by

$$\mathbf{G}^A_{i,j} = \mathbf{R}^{ji}_\theta, \text{ for } i \in [k_A], j \in [n], \text{ and}$$
$$\mathbf{G}^B_{i,j} = \mathbf{R}^{(jk_A)i}_\theta, \text{ for } i \in [k_B], j \in [n].$$

The master node operates according to the encoding rule discussed previously (*cf.* (5.3)) for both $\mathbf{A}$ and $\mathbf{B}$. Thus, each worker node stores $\gamma_A = 1/k_A$ and $\gamma_B = 1/k_B$ fraction of $\mathbf{A}$ and $\mathbf{B}$ respectively. The i-th worker node computes the pairwise product of the matrices $\hat{\mathbf{A}}^T_{\langle i,l_1\rangle}\hat{\mathbf{B}}_{\langle i,l_2\rangle}$ for $l_1, l_2 = 0, 1$ and returns the result to the master node. Thus, the master node needs to recover all pair-wise products of the form $\mathbf{A}^T_{\langle i,\alpha\rangle}\mathbf{B}_{\langle j,\beta\rangle}$ for $i \in [k_A], j \in [k_B]$ and $\alpha, \beta = 0, 1$. Let $\mathbf{Z}$ denote a $1 \times 4k_A k_B$ block matrix that contains all of these pair-wise products.

Theorem 6. The threshold for the rotation matrix based matrix-matrix multiplication scheme is $k_A k_B$. The worst case condition number is bounded by $O(q^{q-k_A k_B+6})$.

Proof. Let $\tau = k_A k_B$ and suppose that the workers indexed by $i_0, \ldots, i_{\tau-1}$ complete their tasks. Let $\mathbf{G}^A_l$ denote the l-th block column of $\mathbf{G}^A$ (with similar notation for $\mathbf{G}^B$). Note that for $k_1, k_2 \in \{0, 1\}$ the l-th worker node computes $\hat{\mathbf{A}}^T_{\langle l,k_1\rangle}\hat{\mathbf{B}}_{\langle l,k_2\rangle}$ which can written as

$$\left(\sum_{\alpha\in[k_A],\beta\in\{0,1\}} \mathbf{G}^A(2\alpha+\beta,2l+k_1)\mathbf{A}^T_{\langle\alpha,\beta\rangle}\right)$$
$$\left(\sum_{\alpha\in[k_B],\beta\in\{0,1\}} \mathbf{G}^B(2\alpha+\beta,2l+k_2)\mathbf{B}_{\langle\alpha,\beta\rangle}\right)$$
$$\equiv \mathbf{Z}\cdot(\mathbf{G}^A(:,2l+k_1)\otimes\mathbf{G}^B(:,2l+k_2)),$$

using the properties of the Kronecker product. Based on this, it can be observed that the decodability of $\mathbf{Z}$ at the master node is equivalent to checking whether the following matrix is full-rank.

$$\tilde{\mathbf{G}} = [\mathbf{G}^A_{i_0}\otimes\mathbf{G}^B_{i_0}|\mathbf{G}^A_{i_1}\otimes\mathbf{G}^B_{i_1}|\ldots|\mathbf{G}^A_{i_{\tau-1}}\otimes\mathbf{G}^B_{i_{\tau-1}}].$$

To analyze this matrix, consider the following decomposition of $\mathbf{G}^A_l\otimes\mathbf{G}^B_l$, for $l\in[n]$.

$$\mathbf{G}^A_l\otimes\mathbf{G}^B_l = \begin{bmatrix} \mathbf{Q}\mathbf{Q}^* \\ \mathbf{Q}\Lambda^l\mathbf{Q}^* \\ \vdots \\ \mathbf{Q}\Lambda^{l(k_A-1)}\mathbf{Q}^* \end{bmatrix} \otimes \begin{bmatrix} \mathbf{Q}\mathbf{Q}^* \\ \mathbf{Q}\Lambda^{lk_A}\mathbf{Q}^* \\ \vdots \\ \mathbf{Q}\Lambda^{lk_A(k_B-1)}\mathbf{Q}^* \end{bmatrix} =$$
$$(\mathbf{I}_{k_A}\otimes\mathbf{Q}) \begin{bmatrix} \mathbf{I} \\ \Lambda^l \\ \vdots \\ \Lambda^{l(k_A-1)} \end{bmatrix} \begin{bmatrix} \mathbf{Q}^* \end{bmatrix} \otimes (\mathbf{I}_{k_B}\otimes\mathbf{Q}) \begin{bmatrix} \mathbf{I} \\ \Lambda^{lk_A} \\ \vdots \\ \Lambda^{lk_A(k_B-1)} \end{bmatrix} \begin{bmatrix} \mathbf{Q}^* \end{bmatrix},$$

where the first equality uses the eigenvalue decomposition of $\mathbf{R}_\theta$. Applying the properties of Kronecker products, this can be simplified as

$$\underbrace{((\mathbf{I}_{k_A}\otimes\mathbf{Q})\otimes(\mathbf{I}_{k_B}\otimes\mathbf{Q}))}_{\tilde{\mathbf{Q}}_1}\times$$
$$\underbrace{\left(\begin{bmatrix} \mathbf{I} \\ \Lambda^l \\ \vdots \\ \Lambda^{l(k_A-1)} \end{bmatrix} \otimes \begin{bmatrix} \mathbf{I} \\ \Lambda^{lk_A} \\ \vdots \\ \Lambda^{lk_A(k_B-1)} \end{bmatrix}\right)}_{\mathbf{X}_l} \underbrace{\left(\begin{bmatrix} \mathbf{Q}^* \end{bmatrix}^{\otimes 2}\right)}_{\tilde{\mathbf{Q}}_2}.$$

Therefore, we can express

$$
\begin{aligned}
&[\mathbf{G}^A_{i_0} \otimes \mathbf{G}^B_{i_0} | \mathbf{G}^A_{i_1} \otimes \mathbf{G}^B_{i_1} | \dots | \mathbf{G}^A_{i_{\tau-1}} \otimes \mathbf{G}^B_{i_{\tau-1}}] \\
&= \tilde{\mathbf{Q}}_1 [\mathbf{X}_{i_0} | \mathbf{X}_{i_1} | \dots | \mathbf{X}_{i_{\tau-1}}] \begin{bmatrix} \tilde{\mathbf{Q}}_2 & 0 & \dots & 0 \\ 0 & \tilde{\mathbf{Q}}_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \tilde{\mathbf{Q}}_2 \end{bmatrix}.
\end{aligned}
$$

Thus, we can conclude that the invertibility and the condition number of $\tilde{\mathbf{G}}$ only depends on $[\mathbf{X}_{i_0} | \mathbf{X}_{i_1} | \dots | \mathbf{X}_{i_{\tau-1}}]$ as the matrices pre- and post- multiplying it are both unitary. The invertibility of $[\mathbf{X}_{i_0} | \mathbf{X}_{i_1} | \dots | \mathbf{X}_{i_{\tau-1}}]$ follows from an application of Claim 15 in the Appendix B. The proof of Claim 15 also shows that upon appropriate permutation, the matrix $[\mathbf{X}_{i_0} | \mathbf{X}_{i_1} | \dots | \mathbf{X}_{i_{\tau-1}}]$ can be expressed as a block-diagonal matrix with four blocks each of size $\tau \times \tau$. Each of these blocks is a Vandermonde matrix with parameters from the set $\{1, \omega_q, \omega_q^2, \dots, \omega_q^{q-1}\}$. Therefore, $[\mathbf{X}_{i_0} | \mathbf{X}_{i_1} | \dots | \mathbf{X}_{i_{\tau-1}}]$ is non-singular and it follows that the threshold of our scheme is $k_A k_B$. An application of Theorem 3 implies that the worst case condition number is at most $O(q^{q-\tau+6})$. □

5.5 Generalized Distributed Matrix Multiplication

In the previous sections, we consider the case that **A** and **B** are partitioned into block columns. In this section, we consider a more general sceanrio where **A** and **B** are partitioned into block columns and and block rows. This construction resembles the entangled polynomial codes of Yu et al. (2020).

We partition the matrices **A** into $2p$ block-rows and $\Delta_A = k_A$ block-columns. We denote $\mathbf{A} = [\mathbf{A}_{(\langle i,l \rangle, j)}]$, $i \in [p], l \in \{0,1\}, j \in [k_A]$, where $\mathbf{A}_{(\langle i,l \rangle, j)}$ denotes the submatrix indexed by the $\langle i, l \rangle$-th block row and the j-th block-column of **A**. Similarly, we partition **B** into $2p$ block-rows and $\Delta_B = k_B$ block-columns. We let $\theta = 2\pi/q$, where $q \geq n > 2k_A k_B p - 1$ (recall that n is the number of worker nodes) is an odd integer.

The encoding in this scenario is somewhat more complicated to express. We simplify this by leveraging the following simple lemma whose proof follows from basic Kronecker product properties (see Appendix B.0.1).

Lemma 4. Suppose that matrices $\mathbf{M}_1$ and $\mathbf{M}_2$ both have ζ rows and the same column dimension. Consider a 2×2 matrix $\mathbf{\Psi} = [\Psi_{i,j}]$, $i = 0, 1, j = 0, 1$. Then

$$\begin{bmatrix} \Psi_{0,0}\mathbf{M}_1 + \Psi_{0,1}\mathbf{M}_2 \\ \Psi_{1,0}\mathbf{M}_1 + \Psi_{1,1}\mathbf{M}_2 \end{bmatrix} = (\mathbf{\Psi} \otimes \mathbf{I}_\zeta) \begin{bmatrix} \mathbf{M}_1 \\ \mathbf{M}_2 \end{bmatrix}$$

Definition 11. Suppose that $\mathbf{A}_{\langle i,l \rangle}$ and $\mathbf{B}_{\langle i,l \rangle}$ have $\zeta = \frac{t}{2p}$ rows. The encoding matrix $\mathbf{A}$ and $\mathbf{B}$ can be denoted as follows.

$$\begin{bmatrix} \hat{\mathbf{A}}_{\langle k,0 \rangle} \\ \hat{\mathbf{A}}_{\langle k,1 \rangle} \end{bmatrix} = \sum_{i=0}^{p-1} \sum_{j=0}^{k_A-1} (\mathbf{R}_{-\theta}^{k((j-1)p+i+1)} \otimes \mathbf{I}_\zeta) \begin{bmatrix} \mathbf{A}_{(\langle i,0 \rangle, j)} \\ \mathbf{A}_{(\langle i,1 \rangle, j)} \end{bmatrix}, \text{ and}$$

$$\begin{bmatrix} \hat{\mathbf{B}}_{\langle k,0 \rangle} \\ \hat{\mathbf{B}}_{\langle k,1 \rangle} \end{bmatrix} = \sum_{i=0}^{p-1} \sum_{j=0}^{k_B-1} (\mathbf{R}_{\theta}^{k(p-1-i+jpk_A)} \otimes \mathbf{I}_\zeta) \begin{bmatrix} \mathbf{B}_{(\langle i,0 \rangle, j)} \\ \mathbf{B}_{(\langle i,1 \rangle, j)} \end{bmatrix}.$$

The k-th worker node stores $\hat{\mathbf{A}}_{\langle k,t \rangle}$, $\hat{\mathbf{B}}_{\langle k,t \rangle}$, $t = 0, 1$. Thus, each worker node stores $\gamma_A = \frac{2}{2pk_A} = \frac{1}{pk_A}$ and $\gamma_B = \frac{2}{2pk_B} = \frac{1}{pk_B}$ fraction of $\mathbf{A}$ and $\mathbf{B}$ respectively. Worker node k computes

$$\begin{bmatrix} \hat{\mathbf{A}}_{\langle k,0 \rangle} \\ \hat{\mathbf{A}}_{\langle k,1 \rangle} \end{bmatrix}^T \begin{bmatrix} \hat{\mathbf{B}}_{\langle k,0 \rangle} \\ \hat{\mathbf{B}}_{\langle k,1 \rangle} \end{bmatrix}. \tag{5.10}$$

Before presenting our decoding algorithm and the main result of this section, we discuss the following example that helps clarify the underlying idea.

Example 19. Suppose $k_A = 1, k_B = 1, p = 2, \ell = 2$. Let $n = 4$. The matrix $\mathbf{A}$ and $\mathbf{B}$ can be partitioned as follows.

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{(\langle 0,0 \rangle,0)} \\ \mathbf{A}_{(\langle 0,1 \rangle,0)} \\ \mathbf{A}_{(\langle 1,0 \rangle,0)} \\ \mathbf{A}_{(\langle 1,1 \rangle,0)} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{B}_{(\langle 0,0 \rangle,0)} \\ \mathbf{B}_{(\langle 0,1 \rangle,0)} \\ \mathbf{B}_{(\langle 1,0 \rangle,0)} \\ \mathbf{B}_{(\langle 1,1 \rangle,0)} \end{bmatrix}.$$

In this example, since $k_A = k_B = 1$, there is only one block column in $\mathbf{A}$ and $\mathbf{B}$. Therefore, the index j in $\mathbf{A}_{(\langle i,l\rangle,j)}$ and $\mathbf{B}_{(\langle i,l\rangle,j)}$ is always 0. Accordingly, to simplify our presentation, we only use indices i and l to refer the respective constituent block rows of $\mathbf{A}$ and $\mathbf{B}$. That is, we simplify $\mathbf{A}_{(\langle i,l\rangle,j)}$ and $\mathbf{B}_{(\langle i,l\rangle,j)}$ to $\mathbf{A}_{\langle i,l\rangle}$ and $\mathbf{B}_{\langle i,l\rangle}$, respectively. Our scheme aims to let the master node recover $\mathbf{A}^T\mathbf{B} = \mathbf{A}^T_{\langle 0,0\rangle}\mathbf{B}_{\langle 0,0\rangle} + \mathbf{A}^T_{\langle 0,1\rangle}\mathbf{B}_{(\langle 0,1\rangle} + \mathbf{A}^T_{\langle 1,0\rangle}\mathbf{B}_{\langle 1,0\rangle} + \mathbf{A}^T_{\langle 1,1\rangle}\mathbf{B}_{\langle 1,1\rangle}$.

Suppose that $\mathbf{A}_{\langle i,l\rangle}$ and $\mathbf{B}_{\langle i,l\rangle}$ have ζ rows. The encoding process can be defined as

$$\begin{bmatrix}\hat{\mathbf{A}}_{\langle k,0\rangle}\\ \hat{\mathbf{A}}_{\langle k,1\rangle}\end{bmatrix} = \sum_{i=0}^{1}(\mathbf{R}_{-\theta}^{k(i-1)}\otimes \mathbf{I}_\zeta)\begin{bmatrix}\mathbf{A}_{\langle i,0\rangle}\\ \mathbf{A}_{\langle i,1\rangle}\end{bmatrix}, \text{ and}$$

$$\begin{bmatrix}\hat{\mathbf{B}}_{\langle k,0\rangle}\\ \hat{\mathbf{B}}_{\langle k,1\rangle}\end{bmatrix} = \sum_{i=0}^{1}(\mathbf{R}_{\theta}^{k(1-i)}\otimes \mathbf{I}_\zeta)\begin{bmatrix}\mathbf{B}_{\langle i,0\rangle}\\ \mathbf{B}_{\langle i,1\rangle}\end{bmatrix}.$$

The computation in worker node k (*cf.* (5.10)) can be analyzed as follows.

Let $\begin{bmatrix}\mathbf{A}^{\mathcal{F}}_{\langle i,0\rangle}\\ \mathbf{A}^{\mathcal{F}}_{\langle i,1\rangle}\end{bmatrix} = (\mathbf{Q}\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{A}_{\langle i,0\rangle}\\ \mathbf{A}_{\langle i,1\rangle}\end{bmatrix}$ and $\begin{bmatrix}\mathbf{B}^{\mathcal{F}}_{\langle i,0\rangle}\\ \mathbf{B}^{\mathcal{F}}_{\langle i,1\rangle}\end{bmatrix} = (\mathbf{Q}\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{B}_{\langle i,0\rangle}\\ \mathbf{B}_{\langle i,1\rangle}\end{bmatrix}$. Then

$$\begin{aligned}
\begin{bmatrix}\hat{\mathbf{A}}_{\langle k,0\rangle}\\ \hat{\mathbf{A}}_{\langle k,1\rangle}\end{bmatrix}^T \begin{bmatrix}\hat{\mathbf{B}}_{\langle k,0\rangle}\\ \hat{\mathbf{B}}_{\langle k,1\rangle}\end{bmatrix} &\overset{(a)}{=} \left((\mathbf{Q}\otimes\mathbf{I}_\zeta)\begin{bmatrix}\hat{\mathbf{A}}_{\langle k,0\rangle}\\ \hat{\mathbf{A}}_{\langle k,1\rangle}\end{bmatrix}\right)^*(\mathbf{Q}\otimes\mathbf{I}_\zeta)\begin{bmatrix}\hat{\mathbf{B}}_{\langle k,0\rangle}\\ \hat{\mathbf{B}}_{\langle k,1\rangle}\end{bmatrix}\\
&= \left((\mathbf{Q}\otimes\mathbf{I}_\zeta)(\mathbf{R}_{-\theta}^{-k}\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{A}_{\langle 0,0\rangle}\\ \mathbf{A}_{\langle 0,1\rangle}\end{bmatrix} + (\mathbf{Q}\otimes\mathbf{I}_\zeta)(\mathbf{I}_2\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{A}_{\langle 1,0\rangle}\\ \mathbf{A}_{\langle 1,1\rangle}\end{bmatrix}\right)^*\\
&\quad\left((\mathbf{Q}\otimes\mathbf{I}_\zeta)(\mathbf{R}_{\theta}^{k}\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{B}_{\langle 0,0\rangle}\\ \mathbf{B}_{\langle 0,1\rangle}\end{bmatrix} + (\mathbf{Q}\otimes\mathbf{I}_\zeta)(\mathbf{I}_2\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{B}_{\langle 1,0\rangle}\\ \mathbf{B}_{\langle 1,1\rangle}\end{bmatrix}\right)\\
&\overset{(b)}{=} \left((\mathbf{Q}\mathbf{R}_{-\theta}^{-k}\mathbf{Q}^*\otimes\mathbf{I}_\zeta)(\mathbf{Q}\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{A}_{\langle 0,0\rangle}\\ \mathbf{A}_{\langle 0,1\rangle}\end{bmatrix} + (\mathbf{Q}\mathbf{I}_2\mathbf{Q}^*\otimes\mathbf{I}_\zeta)(\mathbf{Q}\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{A}_{\langle 1,0\rangle}\\ \mathbf{A}_{\langle 1,1\rangle}\end{bmatrix}\right)^*\\
&\quad\left((\mathbf{Q}\mathbf{R}_{\theta}^{k}\mathbf{Q}^*\otimes\mathbf{I}_\zeta)(\mathbf{Q}\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{B}_{\langle 0,0\rangle}\\ \mathbf{B}_{\langle 0,1\rangle}\end{bmatrix} + (\mathbf{Q}\mathbf{I}_2\mathbf{Q}^*\otimes\mathbf{I}_\zeta)(\mathbf{Q}\otimes\mathbf{I}_\zeta)\begin{bmatrix}\mathbf{B}_{\langle 1,0\rangle}\\ \mathbf{B}_{\langle 1,1\rangle}\end{bmatrix}\right)
\end{aligned}$$

$$
\begin{aligned}
&\overset{(c)}{=}\Bigg(\Big(\begin{bmatrix} \omega_q^{*-k} & \\ & \omega_q^{*k} \end{bmatrix} \otimes \mathbf{I}_\zeta)(\mathbf{Q} \otimes \mathbf{I}_\zeta) \begin{bmatrix} \mathbf{A}_{\langle 0,0\rangle} \\ \mathbf{A}_{\langle 0,1\rangle} \end{bmatrix} + (\begin{bmatrix} 1 & \\ & 1 \end{bmatrix} \otimes \mathbf{I}_\zeta)(\mathbf{Q} \otimes \mathbf{I}_\zeta) \begin{bmatrix} \mathbf{A}_{\langle 1,0\rangle} \\ \mathbf{A}_{\langle 1,1\rangle} \end{bmatrix} \Bigg)^* \\
&\Bigg(\begin{bmatrix} {\omega_q}^{k} & \\ & {\omega_q}^{-k} \end{bmatrix} \otimes \mathbf{I}_\zeta)(\mathbf{Q} \otimes \mathbf{I}_\zeta) \begin{bmatrix} \mathbf{B}_{\langle 0,0\rangle} \\ \mathbf{B}_{\langle 0,1\rangle} \end{bmatrix} + (\begin{bmatrix} 1 & \\ & 1 \end{bmatrix} \otimes \mathbf{I}_\zeta)(\mathbf{Q} \otimes \mathbf{I}_\zeta) \begin{bmatrix} \mathbf{B}_{\langle 1,0\rangle} \\ \mathbf{B}_{\langle 1,1\rangle} \end{bmatrix} \Bigg) \\
&\overset{(d)}{=}\Bigg(\begin{bmatrix} \omega_q^{*-k}\mathbf{A}^{\mathcal{F}}_{\langle 0,0\rangle} \\ \omega_q^{*k}\mathbf{A}^{\mathcal{F}}_{\langle 0,1\rangle} \end{bmatrix} + \begin{bmatrix} \mathbf{A}^{\mathcal{F}}_{\langle 1,0\rangle} \\ \mathbf{A}^{\mathcal{F}}_{\langle 1,1\rangle} \end{bmatrix} \Bigg)^* \Bigg(\begin{bmatrix} {\omega_q}^{k}\mathbf{B}^{\mathcal{F}}_{\langle 0,0\rangle} \\ {\omega_q}^{-k}\mathbf{B}^{\mathcal{F}}_{\langle 0,1\rangle} \end{bmatrix} + \begin{bmatrix} \mathbf{B}^{\mathcal{F}}_{\langle 1,0\rangle} \\ \mathbf{B}^{\mathcal{F}}_{\langle 1,1\rangle} \end{bmatrix} \Bigg) \\
&=(\mathbf{A}^{\mathcal{F}*}_{\langle 0,0\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 1,0\rangle} + \mathbf{A}^{\mathcal{F}*}_{\langle 1,1\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 0,1\rangle})\omega_q^{-k} + \\
&\quad(\mathbf{A}^{\mathcal{F}*}_{\langle 0,0\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 0,0\rangle} + \mathbf{A}^{\mathcal{F}*}_{\langle 1,0\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 1,0\rangle} + \mathbf{A}^{\mathcal{F}*}_{\langle 0,1\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 0,1\rangle} + \mathbf{A}^{\mathcal{F}*}_{\langle 1,1\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 1,1\rangle}) + \\
&\quad(\mathbf{A}^{\mathcal{F}*}_{\langle 1,0\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 0,0\rangle} + \mathbf{A}^{\mathcal{F}*}_{\langle 0,1\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 1,1\rangle})\omega_q^{k}
\end{aligned}
$$

where

- (a) holds because $\mathbf{Q} \otimes \mathbf{I}_\zeta$ is unitary,

- (b) holds by the mixed-product property of Kronecker product. For example

$$
\begin{aligned}
(\mathbf{Q} \otimes \mathbf{I}_\zeta)(\mathbf{R}_{-\theta}^{-k} \otimes \mathbf{I}_\zeta) &= (\mathbf{Q}\mathbf{R}_{-\theta}^{-k}) \otimes \mathbf{I}_\zeta \\
&= (\mathbf{Q}\mathbf{R}_{-\theta}^{-k}\mathbf{Q}^*\mathbf{Q}) \otimes \mathbf{I}_\zeta \\
&= (\mathbf{Q}\mathbf{R}_{-\theta}^{-k}\mathbf{Q}^* \otimes \mathbf{I}_\zeta)(\mathbf{Q} \otimes \mathbf{I}_\zeta).
\end{aligned}
$$

- (c) holds because $\mathbf{Q}\mathbf{R}_\theta\mathbf{Q}^* = \begin{bmatrix} \omega_q & \\ & \omega_q^{-1} \end{bmatrix}$.

- (d) holds by Lemma 4.

Thus, it is clear that whenever master node collects the results of any three distinct worker nodes, it can recover $\mathbf{A}^{\mathcal{F}*}_{\langle 0,0\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 0,0\rangle} + \mathbf{A}^{\mathcal{F}*}_{\langle 1,0\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 1,0\rangle} + \mathbf{A}^{\mathcal{F}*}_{\langle 0,1\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 0,1\rangle} + \mathbf{A}^{\mathcal{F}*}_{\langle 1,1\rangle}\mathbf{B}^{\mathcal{F}}_{\langle 1,1\rangle}$. However, we observe that

$$
\begin{bmatrix} \mathbf{A}^{\mathcal{F}}_{\langle i,0\rangle} \\ \mathbf{A}^{\mathcal{F}}_{\langle i,1\rangle} \end{bmatrix}^* \begin{bmatrix} \mathbf{B}^{\mathcal{F}}_{\langle i,0\rangle} \\ \mathbf{B}^{\mathcal{F}}_{\langle i,1\rangle} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{\langle i,0\rangle} \\ \mathbf{A}_{\langle i,1\rangle} \end{bmatrix}^T \begin{bmatrix} \mathbf{B}_{\langle i,0\rangle} \\ \mathbf{B}_{\langle i,1\rangle} \end{bmatrix}
$$

Thus, we can equivalently recover $\mathbf{A}^T\mathbf{B}$.

Theorem 7. The threshold for scheme in this section is $2pk_Ak_B - 1$. The worst case condition number of the recovery matrices is upper bounded by $O(q^{q-2pk_Ak_B+7})$.

Proof. We proceed in a similar manner as in Example 19. Following the encoding rules (*cf.* Definition 11) and worker computation rules (*cf.* (5.10)), we can analyze the computation in worker k as follows. Let $(\mathbf{Q} \otimes \mathbf{I}_\zeta) \begin{bmatrix} \mathbf{A}_{(\langle i,0\rangle,j)} \\ \mathbf{A}_{(\langle i,1\rangle,j)} \end{bmatrix} = \begin{bmatrix} \mathbf{A}^{\mathcal{F}}_{(\langle i,0\rangle,j)} \\ \mathbf{A}^{\mathcal{F}}_{(\langle i,1\rangle,j)} \end{bmatrix}$ and $(\mathbf{Q} \otimes \mathbf{I}_\zeta) \begin{bmatrix} \mathbf{B}_{(\langle i,0\rangle,j)} \\ \mathbf{B}_{(\langle i,1\rangle,j)} \end{bmatrix} = \begin{bmatrix} \mathbf{B}^{\mathcal{F}}_{(\langle i,0\rangle,j)} \\ \mathbf{B}^{\mathcal{F}}_{(\langle i,1\rangle,j)} \end{bmatrix}$. Then, we have

$$\begin{aligned}
\hat{\mathbf{A}}^{\mathcal{F}}_k &= (\mathbf{Q} \otimes \mathbf{I}_\zeta)\hat{\mathbf{A}}_k = \sum_{i=0}^{p-1}\sum_{j=0}^{k_A-1} (\mathbf{Q}\mathbf{R}_{-\theta}^{k((j-1)p+i+1)}\mathbf{Q}^*\mathbf{Q} \otimes \mathbf{I}_\zeta) \begin{bmatrix} \mathbf{A}_{(\langle i,0\rangle,j)} \\ \mathbf{A}_{(\langle i,1\rangle,j)} \end{bmatrix} \\
&= \sum_{i=0}^{p-1}\sum_{j=0}^{k_A-1} (\Lambda^{*k((j-1)p+i+1)} \otimes \mathbf{I}_\zeta)(\mathbf{Q} \otimes \mathbf{I}_\zeta) \begin{bmatrix} \mathbf{A}_{(\langle i,0\rangle,j)} \\ \mathbf{A}_{(\langle i,1\rangle,j)} \end{bmatrix} \\
&= \begin{bmatrix} \sum_{i=0}^{p-1}\sum_{j=0}^{k_A-1} \omega_q^{*k((j-1)p+i+1)} \mathbf{A}^{\mathcal{F}}_{(\langle i,0\rangle,j)} \\ \sum_{i=0}^{p-1}\sum_{j=0}^{k_A-1} \omega_q^{*-k((j-1)p+i+1)} \mathbf{A}^{\mathcal{F}}_{(\langle i,1\rangle,j)} \end{bmatrix} \\
\hat{\mathbf{B}}^{\mathcal{F}}_k &= (\mathbf{Q} \otimes \mathbf{I}_\zeta)\hat{\mathbf{B}}_k = \sum_{i=0}^{p-1}\sum_{j=0}^{k_B-1} (\mathbf{Q}\mathbf{R}_{\theta}^{k(p-1-i+jpk_A)}\mathbf{Q}^*\mathbf{Q} \otimes \mathbf{I}_\zeta) \begin{bmatrix} \mathbf{B}_{(\langle i,0\rangle,j)} \\ \mathbf{B}_{(\langle i,1\rangle,j)} \end{bmatrix} \\
&= \sum_{i=0}^{p-1}\sum_{j=0}^{k_B-1} (\Lambda^{k(p-1-i+jpk_A)} \otimes \mathbf{I}_\zeta)(\mathbf{Q} \otimes \mathbf{I}_\zeta) \begin{bmatrix} \mathbf{B}^{\mathcal{F}}_{(\langle i,0\rangle,j)} \\ \mathbf{B}^{\mathcal{F}}_{(\langle i,1\rangle,j)} \end{bmatrix} \\
&= \begin{bmatrix} \sum_{i=0}^{p-1}\sum_{j=0}^{k_B-1} \omega_q^{k(p-1-i+jpk_A)} \mathbf{B}^{\mathcal{F}}_{(\langle i,0\rangle,j)} \\ \sum_{i=0}^{p-1}\sum_{j=0}^{k_A-1} \omega_q^{-k(p-1-i+jpk_A)} \mathbf{B}^{\mathcal{F}}_{(\langle i,1\rangle,j)} \end{bmatrix}
\end{aligned}$$

This implies that

$$\begin{aligned}
\hat{\mathbf{A}}_k^T\hat{\mathbf{B}}_k =& ((\mathbf{Q} \otimes \mathbf{I}_\zeta)\hat{\mathbf{A}}_k)^*(\mathbf{Q} \otimes \mathbf{I}_\zeta)\hat{\mathbf{B}}_k \\
=& \hat{\mathbf{A}}^{\mathcal{F}*}_k\hat{\mathbf{B}}^{\mathcal{F}}_k \\
=& \left(\sum_{i=0}^{p-1}\sum_{j=0}^{k_A-1} \omega_q^{k((j-1)p+i+1)} \mathbf{A}^{\mathcal{F}*}_{(\langle i,0\rangle,j)}\right)\left(\sum_{i=0}^{p-1}\sum_{j=0}^{k_B-1} \omega_q^{k(p-1-i+jpk_A)} \mathbf{B}^{\mathcal{F}}_{(\langle i,0\rangle,j)}\right) + \qquad (5.11) \\
& \left(\sum_{i=0}^{p-1}\sum_{j=0}^{k_A-1} \omega_q^{-k((j-1)p+i+1)} \mathbf{A}^{\mathcal{F}*}_{(\langle i,1\rangle,j)}\right)\left(\sum_{i=0}^{p-1}\sum_{j=0}^{k_B-1} \omega_q^{-k(p-1-i+jpk_A)} \mathbf{B}^{\mathcal{F}}_{(\langle i,1\rangle,j)}\right).
\end{aligned}$$

To better understanding the behavior of this sum (5.11), we divide it into two cases,

- *Case 1: Useful terms.* Master node wants to recover $\mathbf{C} = \mathbf{A}^T\mathbf{B} = [\mathbf{C}_{i,j}]$, $i \in [k_A]$, $j \in [k_B]$, where each $\mathbf{C}_{i,j}$ is a block matrix of size $r/k_A \times w/k_B$. Note that $\mathbf{C}_{i,j} = \sum_{u=0}^{p-1}(\mathbf{A}^T_{(\langle u,0\rangle,i)}\mathbf{B}_{(\langle u,0\rangle,j)}+\mathbf{A}^T_{(\langle u,1\rangle,i)}\mathbf{B}_{(\langle u,1\rangle,j)})$. Thus, the "useful" terms in (5.11) are the terms with coefficients $\mathbf{A}^T_{(\langle u,0\rangle,i)}\mathbf{B}_{(\langle u,0\rangle,j)}$, $\mathbf{A}^T_{(\langle u,1\rangle,i)}\mathbf{B}_{(\langle u,1\rangle,j)}$, $u \in [p]$. They correspond to terms $\mathbf{A}^{\mathcal{F}*}_{(\langle u,0\rangle,i)}\mathbf{B}^{\mathcal{F}}_{(\langle u,0\rangle,j)}$ and $\mathbf{A}^{\mathcal{F}*}_{(\langle u,1\rangle,i)}\mathbf{B}^{\mathcal{F}}_{(\langle u,1\rangle,j)}$ in (5.11) since

$$\begin{aligned}
&\mathbf{A}^{\mathcal{F}*}_{(\langle u,0\rangle,i)}\mathbf{B}^{\mathcal{F}}_{(\langle u,0\rangle,j)} + \mathbf{A}^{\mathcal{F}*}_{(\langle u,1\rangle,i)}\mathbf{B}^{\mathcal{F}}_{(\langle u,1\rangle,j)}\\
=&\begin{bmatrix}\mathbf{A}^{\mathcal{F}}_{(\langle u,0\rangle,i)}\\ \mathbf{A}^{\mathcal{F}}_{(\langle u,1\rangle,i)}\end{bmatrix}^* \begin{bmatrix}\mathbf{B}^{\mathcal{F}}_{(\langle u,0\rangle,j)}\\ \mathbf{B}^{\mathcal{F}}_{(\langle u,1\rangle,j)}\end{bmatrix}\\
=&\left(\mathbf{Q}\otimes\mathbf{I}_\zeta\begin{bmatrix}\mathbf{A}_{(\langle u,0\rangle,i)}\\ \mathbf{A}_{(\langle u,1\rangle,i)}\end{bmatrix}\right)^*\left(\mathbf{Q}\otimes\mathbf{I}_\zeta\begin{bmatrix}\mathbf{B}_{(\langle u,0\rangle,j)}\\ \mathbf{B}_{(\langle u,1\rangle,j)}\end{bmatrix}\right)\\
=&\begin{bmatrix}\mathbf{A}_{(\langle u,0\rangle,i)}\\ \mathbf{A}_{(\langle u,1\rangle,i)}\end{bmatrix}^* \begin{bmatrix}\mathbf{B}_{(\langle u,0\rangle,j)}\\ \mathbf{B}_{(\langle u,1\rangle,j)}\end{bmatrix}\\
=&\mathbf{A}^T_{(\langle u,0\rangle,i)}\mathbf{B}_{(\langle u,0\rangle,j)} + \mathbf{A}^T_{(\langle u,1\rangle,i)}\mathbf{B}_{(\langle u,1\rangle,j)}.
\end{aligned}$$

  It is easy to check $\mathbf{A}^{\mathcal{F}*}_{(\langle u,0\rangle,i)}\mathbf{B}^{\mathcal{F}}_{(\langle u,0\rangle,j)}$ is the coefficient of $\omega_q^{k(ip+jpk_A)}$ and $\mathbf{A}^{\mathcal{F}*}_{(\langle u,1\rangle,i)}\mathbf{B}^{\mathcal{F}}_{(\langle u,1\rangle,j)}$ is the coefficient of $\omega_q^{-k(ip+jpk_A)}$.

- *Case 2: Interference terms.* The terms in (5.11) with coefficient $\mathbf{A}^{\mathcal{F}*}_{(\langle u,l\rangle,i)}\mathbf{B}^{\mathcal{F}}_{(\langle v,l\rangle,j)}$ with $u \neq v$ are the *interference terms* and they are the coefficients of $\omega_q^{\pm k(ip+u-v+jpk_A)}$. We conclude that the useful terms have no intersection with interference terms since $|u-v| < p$.

Next we determine the threshold of the proposed scheme. Towards this end, we find the maximum and minimum degree of $\hat{\mathbf{A}}^{\mathcal{F}*}_k\hat{\mathbf{B}}^{\mathcal{F}}_k$ and then argue that (5.11) has consecutive multiples of powers of k. The threshold can then be obtained as the difference between maximum and minimum degree divided by k.

The maximum degree of $\hat{\mathbf{A}}^{\mathcal{F}*}_k\hat{\mathbf{B}}^{\mathcal{F}}_k$ is the degree of term

$$\omega_q^{k(pk_Ak_B-1)}\mathbf{A}^{\mathcal{F}*}_{(\langle p-1,0\rangle,k_A-1)}\mathbf{B}^{\mathcal{F}}_{(\langle 0,0\rangle,k_B-1)},$$

and the minimum degree is term

$$\omega_q^{-k(pk_Ak_B-1)}\mathbf{A}^{\mathcal{F}*}_{(\langle p-1,1\rangle,k_A-1)}\hat{\mathbf{B}}^{\mathcal{F}}_{(\langle 0,1\rangle,k_B-1)}.$$

We observe the power of k in (5.11) can be written as $\pm((j_1-1)p+i_1+1+p-1-i_2+j_2pk_A)=\pm(j_2pk_A+j_1p+i_1-i_2)$, where $j_1\in[k_A], j_2\in[k_B], i_1, i_2\in[p]$. Consider a positive power $d\le pk_Ak_B-1$. We can always find a solution such that $j_2=\lfloor\frac{d}{pk_A}\rfloor$, $j_1=\lfloor\frac{d \bmod pk_A}{p}\rfloor$, $i_1-i_2=(d \bmod pk_A) \bmod p$. The same result can be generalized when d is negative.

Then the threshold of this scheme is $2pk_Ak_B-1$.

(5.11) shows that proposed encoding matrix in transformed domain is a Vandermonde matrix with coefficients $\{\omega_q^{-(pk_Ak_B-1)},\cdots,\omega_q^{pk_Ak_B-1}\}$. An application of Theorem 3 implies that the worst case condition number is upper bounded by $O(q^{q-2pk_Ak_B+7})$. □

The proof of Theorem 7 also illustrates the decoding algorithm. Let i-th worker node compute $\hat{\mathbf{C}}=\hat{\mathbf{A}}_k^T\hat{\mathbf{B}}_k$. Suppose that the master node receives the computation result from any $2pk_Ak_B-1$ worker nodes, which are denoted by $\hat{\mathbf{C}}_{i_0},\cdots,\hat{\mathbf{C}}_{i_{2pk_Ak_B-2}}$. By (5.11), $\mathbf{A}^{\mathcal{F}*}_{(\langle u,l\rangle,i)}\mathbf{B}^{\mathcal{F}}_{(\langle v,l\rangle,j)}$, $u,v\in[p], i\in[k_A], j\in[k_B]$, can be decoded by vector $[\hat{\mathbf{C}}_{i_0},\cdots,\hat{\mathbf{C}}_{i_{2pk_Ak_B-2}}]$ multiplying the inverse of the Vandermonde matrix

$$\begin{bmatrix}
\omega_q^{-i_0(pk_Ak_B-1)} & \omega_q^{-i_1(pk_Ak_B-1)} & \cdots & \omega_q^{-i_{2pk_Ak_B-2}(pk_Ak_B-1)}\\
\omega_q^{-i_0(pk_Ak_B-2)} & \omega_q^{-i_1(pk_Ak_B-2)} & \cdots & \omega_q^{-i_{2pk_Ak_B-2}(pk_Ak_B-2)}\\
\vdots & \vdots & \ddots & \vdots\\
1 & 1 & \cdots & 1\\
\vdots & \vdots & \ddots & \vdots\\
\omega_q^{i_0(pk_Ak_B-1)} & \omega_q^{i_1(pk_Ak_B-1)} & \cdots & \omega_q^{i_{2pk_Ak_B-2}(pk_Ak_B-1)}
\end{bmatrix}$$

Finally, the result $\mathbf{C}=[\mathbf{C}_{i,j}]$, $i\in[k_A]$, $j\in[k_B]$ can be recovered since $\mathbf{C}_{i,j}=\sum_{u=0}^{p-1}(\mathbf{A}^T_{(\langle u,0\rangle,i)}\mathbf{B}_{(\langle u,0\rangle,j)}+\mathbf{A}^T_{(\langle u,1\rangle,i)}\mathbf{B}_{(\langle u,1\rangle,j)})=\sum_{u=0}^{p-1}(\mathbf{A}^{\mathcal{F}*}_{(\langle u,0\rangle,i)}\mathbf{B}^{\mathcal{F}}_{(\langle u,0\rangle,j)}+\mathbf{A}^{\mathcal{F}*}_{(\langle u,1\rangle,i)}\mathbf{B}^{\mathcal{F}}_{(\langle u,1\rangle,j)})$

We compare proposed threshold to Fahim and Cadambe (2019). The threshold of Fahim and Cadambe (2019) is

$$\tau_{M-V}=4k_Ak_Bp-2(k_Ak_B+pk_A+pk_B)+k_A+k_B+2p-1$$

Since when $p = 1$, our threshold in Section 5.4 is better than τ_{M-V}, we only interest in the case $p > 1$. Let $\tau_{dif} = \tau_{M-V} - \tau_{proposed} = 2k_Ak_Bp - 2(k_Ak_B + pk_A + pk_B) + k_A + k_B + 2p$. We show our comparison in Claim 16.

5.6 Comparisons and Numerical Experiments

Suppose that the number of workers n is odd, so that we can pick $q = n$ for the rotation matrix embedding. From a theoretical perspective our schemes have a worst case condition number (over the different recovery submatrices) of $O(q^{q-\tau+6})$ where τ is the recovery threshold. In contrast, the scheme of Yu et al. (2017) has condition numbers that are exponential in n. The work most closely related to ours is by Fahim and Cadambe (2019), which demonstrates an upper bound of $O(q^{2(q-\tau)})$ on the worst case condition number. It can be noted that this grows much faster than our upper bound in the parameter $q - \tau$. In numerical experiments, our worst case condition numbers are much smaller than the work of Fahim and Cadambe (2019); we discuss this in detail below.

Certain approaches Mallick et al. (2019); Das et al. (2018); Das and Ramamoorthy (2019); Ramamoorthy et al. (2019) only apply for matrix-vector multiplication and furthermore do not provide any explicit guarantees on the worst case condition number. Other approaches include the work of Subramaniam et al. (2019) which uses random linear encoding of the **A** and **B** matrices and the work of Das et al. (2019) that uses a convolutional coding approach to this problem. We note that both these approaches work via random sampling and do not have a theoretical upper bound on the worst case condition number. We present numerical experiments comparing our work with Yu et al. (2017); Fahim and Cadambe (2019); Subramaniam et al. (2019) below.

5.6.1 Numerical Experiments

The central point of our paper is that we can leverage the well-conditioned behavior of Vandermonde matrices with parameters on the unit circle while continuing to work with computation over the reals. We compare our results with the work of Yu et al. (2017) (called "Real Vandermonde"),

Table 5.1 Comparison for matrix-vector case with $n = 31$, $\mathbf{A}$ has size 28000×19720 and $\mathbf{x}$ has length 28000.

Scheme	γ_A	τ	Avg. Cond. Num.	Max. Cond. Num.	Avg. Worker Comp. Time (s)	Dec. Time (s)
Real Vand.	1/29	29	1.1×10^{13}	2.9×10^{13}	1.2×10^{-3}	9×10^{-5}
Complex Vand.	1/29	29	12	55	2.9×10^{-3}	2.8×10^{-4}
Circ. Perm. Embed.	1/28	29	12	55	1.2×10^{-3}	3.7×10^{-4}
Rot. Mat. Embed.	1/29	29	12	55	1.3×10^{-3}	10^{-4}

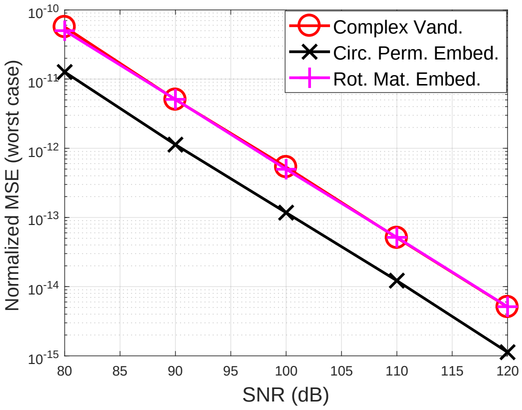

Figure 5.1 Consider matrix-vector $\mathbf{A}^T\mathbf{x}$ multiplication system with $n = 31$, $\tau = 29$. $\mathbf{A}$ has size 28000×19720 and $\mathbf{x}$ has length 28000.

a "Complex Vandermonde" scheme where the evaluation points are chosen from the complex unit circle, the work of Fahim and Cadambe (2019) and Subramaniam et al. (2019).

All experiments were run on the AWS EC2 system with a t2.2xlarge instance (for master node) and t2.micro instances (for slave nodes).

5.6.1.1 Matrix-vector case

In Table 5.1, we compare the *average* and *worst case* condition number of the different schemes for matrix-vector multiplication. The system under consideration has $n = 31$ worker nodes and a threshold specified by the third column (labeled as τ). The evaluation points for Yu et al. (2017) were uniformly sampled from the interval $[-1, 1]$ Berrut and Trefethen (2004). The Complex

Table 5.2 Comparison for $\mathbf{A}^T\mathbf{B}$ matrix-matrix multiplication case with $n = 31$, $k_A = 4$, $k_B = 7$. $\mathbf{A}$ has size 8000×14000, $\mathbf{B}$ has size 8400×14000.

Scheme	γ_A	γ_B	τ	Avg. Cond. Num.	Max. Cond. Num.	Avg. Worker Comp. Time (s)	Dec. Time (s)
Real Vand.	1/4	1/7	28	4.9×10^{12}	2.3×10^{13}	2.132	0.407
Complex Vand.	1/4	1/7	28	27	404	8.421	1.321
Rot. Mat. Embed.	1/4	1/7	28	27	404	2.121	0.408
Fahim and Cadambe (2019)	1/4	1/7	28	1449	8.3×10^4	2.263	0.412
Subramaniam et al. (2019)	1/4	1/7	28	255	5.6×10^4	2.198	0.406

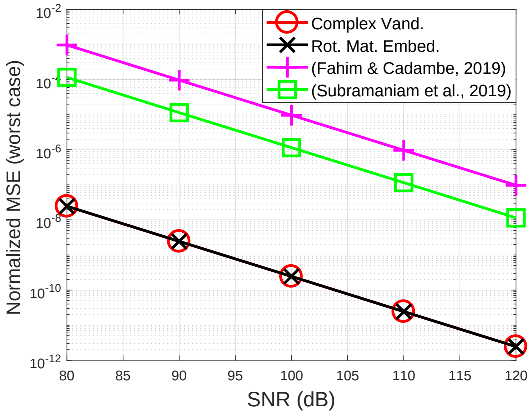

Figure 5.2 Consider matrix-matrix $\mathbf{A}^T\mathbf{B}$ multiplication system with $n = 31$, $k_A = 4$, $k_B = 7$, $\mathbf{A}$ is of size 8000×14000, $\mathbf{B}$ is of 8400×14000.

Vandermonde scheme has evaluation points which are the 31-st root of unity. The Fahim and Cadambe (2019) and Subramaniam et al. (2019) schemes are not applicable for the matrix-vector case. It can be observed from Table 5.1 that the both the worst case and the average condition numbers of our scheme are over eleven orders of magnitude better than the Real Vandermonde scheme. Furthermore, there is an exact match of the condition number values for all the other schemes. This can be understood by following the discussion in Section 5.3. Specifically, our schemes have the property that the condition number only depends on the eigenvalues of corresponding circulation permutation matrix and rotation matrix respectively. These eigenvalues lie precisely within 31-th roots of unity.

It can be observed that the decoding flop count for both matrix-vector and matrix-matrix multiplication is independent of t, i.e., in the regime where t is very large the decoding time may be neglected with respect to the worker node computation time. Nevertheless, from a practical perspective it is useful to understand the decoding times as well.

When the matrix $\mathbf{A}$ is of dimension 28000×19720 and $\mathbf{x}$ is of length 28000, the last two columns in Table 5.1 indicate the average worker node computation time and the master node decoding time for the different schemes. These numbers were obtained by averaging over several runs of the algorithm. It can be observed that the Complex Vandermonde scheme requires about twice the worker computation time as our schemes. Thus, it is wasteful of worker node computation resources. On the other hand, our schemes leverage the same condition number with computation over the reals. The decoding times of almost all the schemes are quite small. However, the Circulant Permutation Matrix scheme requires decoding time which is somewhat higher than the rotation matrix embedding even though we can use FFT based approaches for it. We expect that for much larger scale problems, the FFT based approach may be faster.

Our next set of results compare the mean-squared error (MSE) in the decoded result for the different schemes. Let $\mathbf{A}^T\mathbf{x}$ denote the precise value of the computation and $\widehat{\mathbf{A}^T\mathbf{x}}$ denote the result of using one of the discussed methods. The normalized MSE is defined as $\frac{||\mathbf{A}^T\mathbf{x}-\widehat{\mathbf{A}^T\mathbf{x}}||_2}{||\mathbf{A}^T\mathbf{x}||_2}$. To simulate numerical precision problems, we added i.i.d. Gaussian noise (of different SNRs) to the result of the worker node computation. The master node then performs decoding on the noisy vectors. The plots in Figure 5.1 correspond to the worst case choice of worker nodes for each of the schemes. It can be observed that the Circulant Permutation Matrix Embedding has the best performance. This is because the many of the matrices on the block-diagonal in (B.2) (see Appendix B) have well-behaved condition numbers and only a few correspond to the worst case. We have not shown the results for the Real Vandermonde case here because the normalized MSE was close 1.0.

Table 5.3 Comparison for matrix-matrix $\mathbf{A}^T\mathbf{B}$ multiplication case with $n = 17$, $u_A = 2$, $u_B = 2$, $p = 2$, $\mathbf{A}$ is of size 4000×16000, $\mathbf{B}$ is of 4000×16000.

Scheme	γ_A	γ_B	τ	Avg. Cond. Num.	Max. Cond. Num.	Avg. Worker Comp. Time (s)	Dec. Time (s)
Real Vand.	1/4	1/4	9	1.8×10^4	2×10^6	2.24	0.11
Complex Vand.	1/4	1/4	9	51	1.8×10^3	8.15	0.38
Rot. Mat. Embed.	1/4	1/4	15	7	22	2.23	0.69
Fahim and Cadambe (2019)	1/4	1/4	15	10^4	2.7×10^5	2.23	0.18

5.6.1.2 Matrix-Matrix case

In the matrix-matrix scenario we again consider a system with $n = 31$ worker nodes and $k_A = 4$ and $k_B = 7$ so that the threshold $\tau = k_A k_B = 28$. Once again we observe that the worst case condition number of the Rotation Matrix Embedding is about eleven orders of magnitude lower than the Real Vandermonde case. Furthermore, the schemes of Fahim and Cadambe (2019) and Subramaniam et al. (2019) have a worst case condition numbers that are three orders of magnitude and two orders of magnitude higher than our scheme. For the Subramaniam et al. (2019) scheme we performed 200 random trials and picked the scheme with the lowest worst case condition number.

When the matrix $\mathbf{A}$ is of dimension 8000×14000 and $\mathbf{B}$ is of dimension 8000×14000, the worker node computation times and decoding times are listed in Table 5.2. As expected the Complex Vandermonde scheme takes much longer for the worker node computations, whereas the Rotation Matrix Embedding, Fahim and Cadambe (2019) and Subramaniam et al. (2019) take about the same time. The decoding times are also very similar. For the matrix-matrix case the normalized MSE is defined as $\frac{||\mathbf{A}^T\mathbf{B} - \widehat{\mathbf{A}^T\mathbf{B}}||_F}{||\mathbf{A}^T\mathbf{B}||_F}$ where $\mathbf{A}^T\mathbf{B}$ is the true product and $\widehat{\mathbf{A}^T\mathbf{B}}$ is the decoded product using one of the methods. As shown in Figure 5.2, the normalized MSE of our Rotation Matrix Embedding scheme is much about five orders of magnitude lower than the scheme of Fahim and Cadambe (2019). The normalized MSE of the Real Vandermonde case is almost 1.0 so we do not plot it.

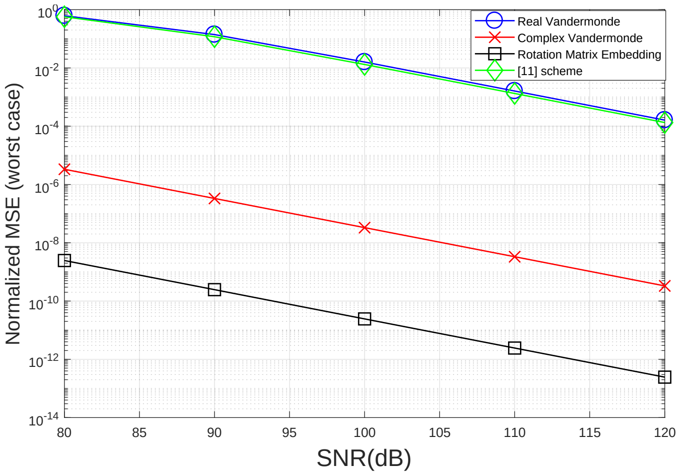

Figure 5.3 Consider matrix-matrix $\mathbf{A}^T\mathbf{B}$ multiplication system with $n = 18$, $u_A = 2$, $u_B = 2$, $p = 2$, $\mathbf{A}$ is of size 4000×16000, $\mathbf{B}$ is of 4000×16000.

In the matrix-matrix multiplication scenario with $p \geq 2$, we consider a system with $n = 17$ worker nodes and $u_A = 2, u_B = 2, p = 2$. We emphasis four schemes in Table 5.3 have different thresholds. The Vandermonde-based schemes have lower threshold than Rotation Matrix Embedding scheme and Fahim and Cadambe (2019) scheme. As we have discussed in Claim 16, in most cases, the Rotation Matrix Embedding scheme has a lower threshold than Fahim and Cadambe (2019) scheme. However, for a fair comparison, we consider a case that they have same thresholds. We observe that Rotation Matrix Embedding scheme has much lower condition number than other three schemes. Notice that it is even lower than Complex Vandermonde scheme, which is different from matrix-vector and matrix-matrix multiplication. It is because in Complex Vandermonde scheme with $p \geq 2$, its encoding matrix is a Vandermonde matrix of size 9×17, whose generators are roots of unity. Then by Claim 3, the condition number is bounded by $O(17^{14})$. The encoding matrix of Rotation Matrix Embedding scheme is a Vandermonde matrix of size 15×17, whose condition number is upper bounded by $O(17^{8})$. As shown in Figure 5.3, the normalized MSE of our Rotation Matrix Embedding scheme is much lower than the other schemes. As for the computation and decoding time in Table 5.3, the Complex Vandermonde scheme requires higher computation and decoding time than the Real Vandermonde scheme (4x) since they are operated over the complex field. Fahim and Cadambe (2019) requires higher decoding time than the Real Vandermonde scheme (2x) since its threshold is higher. But they have same computation time. The Rotation Matrix Embedding scheme has highest decoding time since it have higher threshold than Real/Complex Vandermonde scheme decoding algorithm are operated over the complex field.

5.6.1.3 Matrix-Matrix case with finite field embedding

We now consider the finite field embedding proposed in Yu et al. (2020). As discussed before this is mentioned as a potential solution to the numerical issues encountered when operating over the reals in Section VII of Yu et al. (2020). For this purpose the real entries will need to multiplied by large enough integers and then quantized so that each entry lies with 0 and $p-1$ for a large enough prime p. All computations will be performed within the finite field of order p, i.e., by reducing the computations modulo-p. This technique requires that each $\mathbf{A}_i^T\mathbf{B}_j$ needs to have all its entries within 0 to $p-1$, otherwise there will be errors in the computation.

Let α be an upper bound on the absolute value of matrix entries in $\mathbf{A}$ and $\mathbf{B}$. Then, this means that the following dynamic range constraint (DRC)

$$\alpha^2 t \leq p-1,$$

needs to be satisfied. Otherwise, the modulo-p operation will cause arbitrarily large errors.

We note here that the publicly available code for Yu et al. (2017) uses $p = 65537$. Now consider a system with $k_A = 3$, $k_B = 2$. Even for small matrices with $\mathbf{A}$ of size 400×200, $\mathbf{B}$ of size 400×300 and entries chosen as random integers between 0 to 30, the DRC is violated for $p = 65537$ since $30^2 \times 400 > 65537$. In this scenario, the normalized MSE of the Yu et al. (2017) approach is 0.7746. In contrast, our method has a normalized MSE $\approx 2 \times 10^{-28}$ for the same system with $k_A = 3, k_B = 2$.

When working over 64-bit integers, the largest integer is $\approx 10^{19}$. Thus, even if $t < 10^5$, the method can only support $\alpha \leq 10^7$. Thus, the range is rather limited. Furthermore, considering matrices of limited dynamic range is not a valid assumption. In machine learning scenarios such as deep neural networks, matrix multiplications are applied repeatedly, and the output of one stage serves as the input for the other. Thus, over several iterations the dynamic range of the matrix entries will grow. Thus, applying the finite field embedding technique will necessarily incur quantization error.

The most serious limitation of the method comes from the fact the error in the computation (owing to quantization) is very strongly dependent on the actual entries of the $\mathbf{A}$ and $\mathbf{B}$ matrices. We demonstrate this next. In fact, we can generate structured integer matrices $\mathbf{A}$ and $\mathbf{B}$ such that the normalized MSE of their approach is exactly 1.0. Towards this end we first pick the prime $p = 2147483647$ (which is much larger than their publicly available code) so that their method can support higher dynamic range. Next let $r = w = t = 400$. This implies that $\alpha \leq 1000$ by the dynamic range constraint.

For $k_A = k_B = 2$, the matrices have the following block decomposition.

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}, \text{ and}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix}.$$

Each $\mathbf{A}_{ij}$ and $\mathbf{B}_{ij}$ is a matrix of size 200×200, with entries chosen from the following distributions. $\mathbf{A}_{11}$, $\mathbf{A}_{12}$ distributed Unif$(0, \dots, 9999)$ and $\mathbf{A}_{21}$, $\mathbf{A}_{22}$ distributed Unif$(0, \dots, 9)$. Next, $\mathbf{B}_{11}$, $\mathbf{B}_{12}$ distributed Unif$(0, \dots, 9)$ and $\mathbf{B}_{21}, \mathbf{B}_{22}$ distributed Unif$(0, \dots, 9999)$. In this scenario, the DRC requires us to multiply each matrix by 0.1 and quantize each entry between 0 and 999. Note that this implies that $\mathbf{A}_{21}, \mathbf{A}_{22}, \mathbf{B}_{11}, \mathbf{B}_{12}$ are all quantized into zero submatrices since the entry in these four submatrices is less than 10. We emphasize that the finite field embedding technique *only* recovers the product of these quantized matrices. However, this product is

$$\tilde{\mathbf{A}}^T\tilde{\mathbf{B}} = \begin{bmatrix} \tilde{\mathbf{A}}_{11} & \tilde{\mathbf{A}}_{12} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}^T \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \tilde{\mathbf{B}}_{21} & \tilde{\mathbf{A}}_{22} \end{bmatrix} = \mathbf{0}.$$

Thus, the final estimate of the original product $\mathbf{A}^T\mathbf{B}$, denoted as $\widehat{\mathbf{A}^T\mathbf{B}}$ is the all-zeros matrix. This implies that the normalized MSE of their scheme is exactly 1.0. Thus, the finite field embedding

Table 5.4 Performance of matrix inversion over a large prime order field in Python 3.7. The table shows the computation time for inverting a $\ell \times \ell$ matrix $\mathbf{G}$ over a finite field of order p. Let $\widehat{\mathbf{G}^{-1}}$ denote the inverse obtained by applying the sympy function $Matrix(\mathbf{G}).inverse\_mod(p)$. The MSE is defined as $\frac{1}{\ell}||\mathbf{G}\widehat{\mathbf{G}^{-1}} - \mathbf{I}||_F$.

ℓ	p	Computation Time (s)	MSE
9	65537	1.39	0
12	65537	4.38	0
15	65537	12.64	0
9	2147483647	1.39	0
12	2147483647	4.68	1.8×10^9
15	2147483647	14.45	4.2×10^9

technique has a very strong dependence on the matrix entries. We note here that even if we consider other quantization schemes or larger 64-bit primes, one can arrive at adversarial examples such as the ones shown above. Once again for these examples, our methods have a normalized MSE of at most 10^{-27}.

In our experience, the finite field embedding technique also suffers from significant computational issues in implementation. Note that the technique requires the computation of the inverse matrix at the master node that is required for decoding the final result. We implemented this within the Python 3.7, sympy library (see Tang (2020) Git hub repository). We performed experiments with $p = 65537$ and $p = 2147483647$. As shown in Table 5.4, for the smaller prime $p = 65537$, the inverse computation is accurate up to 15×15 matrices; however, the computation time of the inverse is rather high and can dominate the overall execution time. On the other hand for the larger prime $p = 2147483647$, the error in in the computed inverse is very high for 12×12 and 15×15 matrices; the corresponding time taken is even higher. It is possible that very careful implementations can perhaps avoid these issues. However, we are unaware of any such publicly available code.

To summarize, the finite field embedding technique suffers from major dynamic range limitations and associated computational issues and cannot be used to support real computations.

BIBLIOGRAPHY

([Online] Available: https://bitbucket.org/kkonstantinidis/stragglermitmm/src/master, 2019). Repository of Erasure coding for distributed matrix multiplication for matrices with bounded entries.

([Online] Available: https://github.com/AvestimehrResearchGroup/Polynomial-Code, 2017). Repository of Polynomial Code for prior implementation.

Ahlswede, R., Cai, N., Li, S.-Y., and Yeung, R. W. (2000). Network information flow. *IEEE Trans. on Inf. Theory*, 46(4):1204–1216.

Alon, N., Moitra, A., and Sudakov, B. (2012). Nearly complete graphs decomposable into large induced matchings and their applications. In *Proc. of the 44-th Annual ACM symposium on Theory of computing*, pages 1079–1090.

Baranyai, Z. (1975). On the factorization of the complete uniform hypergraph. In *Infinite and finite sets (Colloq., Keszthely, 1973; dedicated to P. Erdos on his 60th birthday)*, pages 91–108.

Berrut, J.-P. and Trefethen, L. N. (2004). Barycentric lagrange interpolation. *SIAM review*, 46(3):501–517.

Blake, I. F. (1972). Codes over certain rings. *Information and Control*, 20(4):396–404.

Das, A. B. and Ramamoorthy, A. (2019). Distributed matrix-vector multiplication: A convolutional coding approach. In *IEEE Int. Symp. on Inf. Theory*, pages 3022–3026.

Das, A. B., Ramamoorthy, A., and Vaswani, N. (2019). Random convolutional coding for robust and straggler resilient distributed matrix computation. *[Online] Available at: https://arxiv.org/abs/1907.08064*.

Das, A. B., Tang, L., and Ramamoorthy, A. (2018). C^3LES : Codes for coded computation that leverage stragglers. In *IEEE Inf. Th. Workshop*, pages 1–5.

Dougherty, R., Freiling, C., and Zeger, K. (2005). Insufficiency of linear coding in network information flow. *IEEE Trans. on Inf. Theory*, 51(8):2745–2759.

Dummit, D. S. and Foote, R. M. (2003). *Abstract algebra*. Wiley, 3rd Ed.

Dutta, S., Cadambe, V., and Grover, P. (2016). Short-dot: Computing large linear transforms distributedly using coded short dot products. In *Proc. of Adv. in Neural Inf. Proc. Sys.*, pages 2100–2108.

Dutta, S., Fahim, M., Haddadpour, F., Jeong, H., Cadambe, V., and Grover, P. (2019). On the optimal recovery threshold of coded matrix multiplication. *IEEE Trans. on Inf. Theory*, 66(1):278–301.

Fahim, M. and Cadambe, V. R. (2019). Numerically stable polynomially coded computing. *[Online] Available at: https://arxiv.org/abs/1903.08326*.

Fitzpatrick, M. E. ([Online] `Available:http://cdaweb01.storage.toshiba.com/docs/services-support-documents/toshiba_4kwhitepaper.pdf`, 2011). 4K Sector Disk Drives: Transitioning to the Future with Advanced Format Technologies. Toshiba 4K White Paper.

Gautschi, W. (1990). How (un) stable are Vandermonde systems? *Asymptotic and Computational Analysis (Lecture Notes in Pure and Applied Mathematics)*, 124:193–210.

Ghasemi, H. and Ramamoorthy, A. (2016). Further results on lower bounds for coded caching. In *IEEE Int. Symp. on Inf. Theory*, pages 2319–2323.

Ghasemi, H. and Ramamoorthy, A. (2017a). Algorithms for asynchronous coded caching. In *Asilomar Conference on Signals, Systems, and Computers*, pages 636–640.

Ghasemi, H. and Ramamoorthy, A. (2017b). Asynchronous coded caching. In *IEEE Int. Symp. on Inf. Theory*, pages 2438–2442.

Ghasemi, H. and Ramamoorthy, A. (2017c). Improved lower bounds for coded caching. *IEEE Trans. on Inf. Theory*, 63(7):4388–4413.

Graham, R. L., Knuth, D. E., and Patashnik, O. (1994). *Concrete mathematics: a foundation for computer science (2nd ed.)*. Addison-Wesley Professional.

Gray, R. M. (2006). Toeplitz and circulant matrices: A review. *Foundations and Trends® in Communications and Information Theory*, 2(3):155–239.

Higham, N. J. (2002). *Accuracy and Stability of Numerical Algorithms*. SIAM:Society for Industrial and Applied Mathematics.

Ho, T., Médard, M., Koetter, R., Karger, D. R., Effros, M., Shi, J., and Leong, B. (2006). A random linear network coding approach to multicast. *IEEE Trans. on Inf. Theory*, 52(10):4413–4430.

Horn, R. A. and Johnson, C. R. (1991). *Topics in matrix analysis*. Cambridge University Press.

Huang, Q., Tang, L., He, S., Xiong, Z., and Wang, Z. (2014). Low-complexity encoding of quasi-cyclic codes based on galois fourier transform. *IEEE Trans. on Comm.*, 62(6):1757–1767.

Huang, S. and Ramamoorthy, A. (2013). An achievable region for the double unicast problem based on a minimum cut analysis. *IEEE Trans. on Comm.*, 61(7):2890–2899.

Huang, S. and Ramamoorthy, A. (2014). On the multiple unicast capacity of 3-source, 3-terminal directed acyclic networks. *IEEE/ACM Trans. Netw.*, 22(1):285–299.

Huang, S., Ramamoorthy, A., and Medard, M. (2011). Minimum cost mirror sites using network coding: Replication versus coding at the source nodes. *IEEE Trans. on Inf. Theory*, 57(2):1080–1091.

Ji, M., Tulino, A., Llorca, J., and Caire, G. (2015a). Caching-aided coded multicasting with multiple random requests. In *IEEE Inf. Th. Workshop*, pages 1–5.

Ji, M., Wong, M. F., Tulino, A. M., Llorca, J., Caire, G., Effros, M., and Langberg, M. (2015b). On the fundamental limits of caching in combination networks. In *IEEE 16th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 695–699.

Konstantinidis, K. and Ramamoorthy, A. (2018). Leveraging coding techniques for speeding up distributed computing. In *Proc. of IEEE Global Communications Conference*, pages 1–6.

Konstantinidis, K. and Ramamoorthy, A. (2019). Camr: Coded aggregated mapreduce. In *IEEE Int. Symp. on Inf. Theory*, pages 1427–1431.

Konstantinidis, K. and Ramamoorthy, A. (2020 (to appear)). Resolvable designs for speeding up distributed computing. *IEEE/ACM Trans. Netw.*

Langberg, M. and Ramamoorthy, A. (2009). Communicating the sum of sources in a 3-sources/3-terminals network. In *IEEE Int. Symp. on Inf. Theory*, pages 2121–2125.

Lee, K., Lam, M., Pedarsani, R., Papailiopoulos, D., and Ramchandran, K. (2018). Speeding up distributed machine learning using codes. *IEEE Trans. on Inf. Theory*, 64(3):1514–1529.

Lee, K., Suh, C., and Ramchandran, K. (2017). High-dimensional coded matrix multiplication. In *IEEE Int. Symp. on Inf. Theory*, pages 2418–2422.

Li, S., Maddah-Ali, M. A., Yu, Q., and Avestimehr, A. S. (2017). A fundamental tradeoff between computation and communication in distributed computing. *IEEE Trans. on Inf. Theory*, 64(1):109–128.

Li, S.-Y., Yeung, R. W., and Cai, N. (2003). Linear network coding. *IEEE Trans. on Inf. Theory*, 49(2):371–381.

Lin, S. and Costello, D. J. (2004). *Error Control Coding, 2nd Ed.* Prentice Hall.

Maddah-Ali, M. A. and Niesen, U. (2014a). Decentralized coded caching attains order-optimal memory-rate tradeoff. *IEEE/ACM Trans. Netw.*, 23(4):1029–1040.

Maddah-Ali, M. A. and Niesen, U. (2014b). Fundamental limits of caching. *IEEE Trans. on Info. Theory*, 60(5):2856–2867.

Mallick, A., Chaudhari, M., Sheth, U., Palanikumar, G., and Joshi, G. (2019). Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(3):1–40.

Ngai, C. K. and Yeung, R. W. (2004). Network coding gain of combination networks. In *IEEE Inf. Th. Workshop*, pages 283–287.

Olmez, O. and Ramamoorthy, A. (2016). Fractional repetition codes with flexible repair from combinatorial designs. *IEEE Trans. on Inf. Theory*, 62(4):1565 –1591.

Pan, V. (2016). How bad are vandermonde matrices? *SIAM Journal on Matrix Analysis and Applications*, 37(2):676–694.

Pan, V. Y. (2013). *Polynomial Evaluation and Interpolation: Fast and Stable Approximate Solution.* Citeseer.

Rai, B. K. and Dey, B. K. (2012). On network coding for sum-networks. *IEEE Trans. on Inf. Theory*, 58(1):50–63.

Ramamoorthy, A., Das, A. B., and Tang, L. (2020). Straggler-resistant distributed matrix computation via coding theory: Removing a bottleneck in large-scale data processing. *IEEE Signal Processing Magazine*, 37(3):136–145.

Ramamoorthy, A. and Langberg, M. (2013). Communicating the sum of sources over a network. *IEEE Journal on Selected Areas in Communications*, 31(4):655–665.

Ramamoorthy, A. and Tang, L. (2019). Numerically stable coded matrix computations via circulant and rotation matrix embeddings. *[Online] Available at: https://arxiv.org/abs/1910.06515.*

Ramamoorthy, A., Tang, L., and Vontobel, P. O. (2019). Universally decodable matrices for distributed matrix-vector multiplication. In *IEEE Int. Symp. on Inf. Theory*, pages 1777–1781.

Roth, R. M. (2006). *Introduction to Coding Theory.* Cambridge University Press.

Shangguan, C., Zhang, Y., and Ge, G. (2018). Centralized coded caching schemes: A hypergraph theoretical approach. *IEEE Trans. on Inf. Theory*, 64(8):5755–5766.

Shanmugam, K., Ji, M., Tulino, A. M., Llorca, J., and Dimakis, A. G. (2014). Finite length analysis of caching-aided coded multicasting. In *52nd Annual Allerton Conference on Communication, Control, and Computing*, pages 914–920.

Shanmugam, K., Ji, M., Tulino, A. M., Llorca, J., and Dimakis., A. G. (2016). Finite-length analysis of caching-aided coded multicasting. *IEEE Trans. on Inf. Theory*, 62(10):5524–5537.

Shanmugam, K., Tulino, A. M., and Dimakis, A. G. (2017). Coded caching with linear subpacketization is possible using Ruzsa-Szeméredi graphs. In *IEEE Int. Symp. on Inf. Theory*, pages 1237–1241.

Stinson, D. R. (2003). *Combinatorial Designs: Construction and Analysis*. Springer.

Subramaniam, A. M., Heidarzadeh, A., and Narayanan, K. R. (2019). Random khatri-rao-product codes for numerically-stable distributed matrix multiplication. In *Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 253–259.

Tang, L. ([Online] Available: https://github.com/litangsky/inverseoverfield, 2020). Github repository for computing matrix inverse over prime order finite field.

Tang, L., Huang, Q., Wang, Z., and Xiong, Z. (2013). Low-complexity encoding of binary quasi-cyclic codes based on galois fourier transform. In *IEEE Int. Symp. on Inf. Theory*, pages 131–135.

Tang, L., Konstantinidis, K., and Ramamoorthy, A. (2019). Erasure coding for distributed matrix multiplication for matrices with bounded entries. *IEEE Comm. Lett.*, 23(1):8–11.

Tang, L. and Ramamoorthy, A. (2016a). Coded caching for networks with the resolvability property. In *IEEE Int. Symp. on Inf. Theory*, pages 420–424.

Tang, L. and Ramamoorthy, A. (2016b). Coded caching with low subpacketization levels. In *Workshop on Network Coding (NetCod)*, pages 1–6.

Tang, L. and Ramamoorthy, A. (2017). Low subpacketization schemes for coded caching. In *IEEE Int. Symp. on Inf. Theory*, pages 2790–2794.

Tang, L. and Ramamoorthy, A. (2018). Coded caching schemes with reduced subpacketization from linear block codes. *IEEE Trans. on Inf. Theory*, 64(4):3099–3120.

Tripathy, A. S. and Ramamoorthy, A. (2015). Capacity of sum-networks for different message alphabets. In *IEEE Int. Symp. on Inf. Theory*, pages 606–610.

Tripathy, A. S. and Ramamoorthy, A. (2017). Sum-networks from incidence structures: construction and capacity analysis. *IEEE Trans. on Inf. Theory*, 64(5):3461–3480.

Wang, S., Liu, J., and Shroff, N. B. (2018). Coded sparse matrix multiplication. In *Proc. 35th Int. Conf. on Mach. Learning*, pages 5139–5147.

Yagle, A. E. (1995). Fast algorithms for matrix multiplication using pseudo-number-theoretic transforms. *IEEE Trans. on Sig. Proc.*, 43(1):71–76.

Yan, Q., Cheng, M., Tang, X., and Chen, Q. (2017a). On the placement delivery array design for centralized coded caching scheme. *IEEE Trans. on Inf. Theory*, 63(9):5821–5833.

Yan, Q., Tang, X., Chen, Q., and Cheng, M. (2017b). Placement delivery array design through strong edge coloring of bipartite graphs. *IEEE Comm. Lett.*, 22(2):236–239.

Yu, Q., Maddah-Ali, M. A., and Avestimehr, A. S. (2017). Polynomial codes: an optimal design for high-dimensional coded matrix multiplication. In *Proc. of Adv. in Neural Inf. Proc. Sys.*, pages 4403–4413.

Yu, Q., Maddah-Ali, M. A., and Avestimehr, A. S. (2018). Characterizing the rate-memory tradeoff in cache networks within a factor of 2. *IEEE Trans. on Inf. Theory*, 65(1):647–663.

Yu, Q., Maddah-Ali, M. A., and Avestimehr, A. S. (2020). Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding. *IEEE Trans. on Inf. Theory*, 66(3):1920–1933.

Zhang, M., Tang, L., Huang, Q., and Wang, Z. (2014). Low complexity encoding algorithm of rs-based qc-ldpc codes. In *IEEE Inf. Th. Workshop*, pages 1–4.

APPENDIX A. SUPPLEMENT FOR CODED CACHING SCHEMES WITH REDUCED SUBPACKETIZATION FROM LINEAR BLOCK CODES

Resolvable design over Z mod q

Lemma 5. A (n, k) linear block code over $\mathbb{Z}$ mod q with generator matrix $\mathbf{G} = [g_{ab}]$ can construct a resolvable block design by the procedure in Section 3.2.1 if $\gcd(q, g_{0b}, g_{1b}, \cdots, g_{(k-1)b}) = 1$ for $0 \leq b < n$.

Proof. Assume $q = q_1 \times q_2 \times \cdots \times q_d$ where q_i, $1 \leq i \leq d$ is a prime or a prime power. If the $\gcd(q, g_{0b}, g_{1b}, \cdots, g_{(k-1)b}) = 1$, then it is evident that $\gcd(q_i, g_{0b}, g_{1b}, \cdots, g_{(k-1)b}) = 1$ for $1 \leq i \leq d$. As q_i is either a prime or a prime power, it follows that there exists a g_{a^*b} which is relatively prime to q_i, i.e., g_{a^*b} is a unit in the ring $\mathbb{Z}$ mod q_i.

Note that for $\Delta = [\Delta_0\ \Delta_1\ \ldots\ \Delta_{n-1}] = \mathbf{u}\mathbf{G}$, we have

$$\Delta_b = \sum_{a=0}^{k-1} \mathbf{u}_a g_{ab}, \tag{A.1}$$

where $\mathbf{u} = [\mathbf{u}_0, \cdots, \mathbf{u}_{k-1}]$. We consider eq. (A.1) over the ring $\mathbb{Z}$ mod q_i and rewrite eq. (A.1) as

$$\Delta_b - \mathbf{u}_{a^*} g_{a^*b} = \sum_{a \neq a^*} \mathbf{u}_a g_{ab},$$

For arbitrary $\mathbf{u}_a$, $a \neq a^*$, this equation has a unique solution for $\mathbf{u}_{a^*}$ since g_{a^*b} is a unit in $\mathbb{Z}$ mod q_i. This implies that there are q_i^{k-1} distinct solutions for (A.1) over $\mathbb{Z}$ mod q_i. Using the Chinese remainder theorem, eq. (A.1) has $q_1^{k-1} \times q_2^{k-1} \times \cdots q_d^{k-1} = q^{k-1}$ solutions over $\mathbb{Z}$ mod q and the result follows. □

Remark 12. From Lemma 5, it can be easily verified that a linear block code over $\mathbb{Z}$ mod q can construct a resolvable block design if one of the following conditions for each column $\mathbf{g}_i$ of the generator matrix is satisfied.

- At least one non-zero entry of $\mathbf{g}_i$ is a unit in $\mathbb{Z} \mod q$, or

- all non-zero entries in $\mathbf{g}_i$ are zero divisors but their greatest common divisor is 1.

For the SPC code over $\mathbb{Z}$ mod q, all the non-zero entries in the generator matrix are 1, which is an unit. Therefore, the construction always results in a resolvable design.

Proof of Lemma 2

First, we show that the proposed delivery scheme allows each user's demand to be satisfied. Note that Claim 2 shows that each user in a parallel class that belongs to the recovery set $\mathcal{S}_a$ recovers all missing subfiles with a specified superscript from it. Thus, we only need to show that if signals are generated (according to Claim 2) for each recovery set, we are done. This is equivalent to showing that the bipartite recovery set graph is such that each parallel class has degree z and multiple edges between nodes are disallowed.

Towards this end, consider the parallel class and we claim that there exist exactly z solutions (a_α, b_α) for integer values of $\alpha = 1, \ldots, z$ to the equation

$$a_\alpha(k+1) + b_\alpha = j + n(\alpha - 1) \tag{A.2}$$

such that $a_{\alpha_1} \neq a_{\alpha_2}$ for $\alpha_1 \neq \alpha_2$ and $j < n$. The existence of the solution for each equation above follows from the division algorithm. Note that $a_\alpha < nz/(k+1)$ as the $RHS < nz$. Furthermore, note that for $1 \leq \alpha_1 \leq z$ and $1 \leq \alpha_2 \leq z$, we cannot have solutions to eq. (A.2) such that $a_{\alpha_1} = a_{\alpha_2}$ as this would imply that $|b_{\alpha_1} - b_{\alpha_2}| \geq n$ which is a contradiction. This shows that each parallel class P_j participates in at least z different recovery sets.

The following facts follow easily from the construction of the recovery sets. The degree of each recovery set in the bipartite graph is $k+1$ and there are $\frac{nz}{k+1}$ of them; multiple edges between a recovery set and a parallel class are disallowed. Therefore, the total number of edges in the bipartite graph is nz. As each parallel class participates in at least z recovery sets, by this argument, it participates in exactly z recovery sets. .

Finally, we calculate the rate of the delivery phase. In total, the server transmits $q^k(q-1)\frac{zn}{k+1}$ equations, where the symbol transmitted has the size of a subfile. Thus, the rate is

$$\begin{aligned} R &= q^k(q-1)\frac{zn}{k+1}\frac{1}{q^k z} \\ &= \frac{(q-1)n}{k+1}. \end{aligned}$$

Proof of Claim 4

The matrix $\mathbf{G}_{\mathcal{S}}$ is shown below.

$$\begin{bmatrix} g_0 & g_1 & \cdot & \cdot & g_{\lceil \frac{k}{2} \rceil -1} & 0 & \cdot & \cdot & 0 & 0 \\ 0 & g_0 & g_1 & \cdot & g_{\lceil \frac{k}{2} \rceil -2} & 0 & \cdot & \cdot & 0 & 0 \\ \vdots & & & & & & & & & \vdots \\ 0 & \cdot & 0 & g_0 & g_1 & 0 & \cdot & \cdot & 0 & 0 \\ 0 & \cdot & \cdot & 0 & g_0 & g_{n-k} & 0 & \cdot & \cdot & 0 \\ 0 & \cdot & \cdot & 0 & 0 & g_{n-k-1} & g_{n-k} & 0 & \cdot & 0 \\ \vdots & & & & & & & & & \vdots \\ 0 & \cdot & \cdot & 0 & 0 & g_{n-k-\lfloor \frac{k}{2} \rfloor} & \cdot & \cdot & \cdot & g_{n-k} \end{bmatrix}.$$

In the above expression and the subsequent discussion if i is such that $i < 0$, we set $g_i = 0$.

By Claim 3, a cyclic code with generator matrix $\mathbf{G}$ satisfies the CCP if all submatrices

$$\begin{aligned} \mathbf{G}_{\mathcal{S}\backslash(a+j)_n} =& [\mathbf{g}_{(a)_n}, \mathbf{g}_{(a+1)_n}, \cdots, \mathbf{g}_{(a+j-1)_n}, \\ & \mathbf{g}_{(a+j+1)_n}, \cdots, \mathbf{g}_{(a+k)_n}], \end{aligned}$$

where $a = n - \lfloor \frac{k}{2} \rfloor - 1$, $0 \leq j \leq k$, have full rank. In what follows, we argue that this is true. Note that in the generator matrix of cyclic code, any k consecutive columns are linearly independent Lin and Costello (2004). Therefore for $j = 0$ and k, $\mathbf{G}_{\mathcal{S}\backslash(a+j)_n}$ has full rank, without needing the conditions of Claim 4. For $0 < j \leq \lfloor \frac{k}{2} \rfloor$, $\mathbf{G}_{\mathcal{S}\setminus(a+j)_n}$ is as below.

$$\begin{bmatrix}
g_0 & g_1 & \cdot & g_{\lceil \frac{k}{2} \rceil -1} & 0 & \cdot & \cdot & \cdot & \cdot & 0 \\
0 & g_0 & \cdot & g_{\lceil \frac{k}{2} \rceil -2} & 0 & \cdot & \cdot & \cdot & \cdot & 0 \\
\vdots & & & & & & & & & \vdots \\
0 & \cdot & g_0 & g_1 & 0 & \cdot & \cdot & \cdot & \cdot & 0 \\
0 & \cdot & 0 & g_0 & g_{n-k} & 0 & \cdot & \cdot & \cdot & 0 \\
0 & \cdot & \cdot & 0 & g_{n-k-1} & g_{n-k} & 0 & \cdot & \cdot & 0 \\
\vdots & & & & & & & & & \vdots \\
0 & \cdot & \cdot & 0 & g_{n-k-j+1} & \cdot & g_{n-k} & 0 & \cdot & 0 \\
0 & \cdot & \cdot & 0 & g_{n-k-j} & \cdot & g_{n-k-1} & 0 & \cdot & 0 \\
0 & \cdot & \cdot & 0 & g_{n-k-j-1} & \cdot & g_{n-k-2} & g_{n-k} & \cdot & 0 \\
\vdots & & & & & & & & & \vdots \\
0 & \cdot & \cdot & 0 & g_{n-k-\lfloor \frac{k}{2} \rfloor} & \cdot & g_{n-k-\lfloor \frac{k}{2} \rfloor + j - 1} & g_{n-k-\lfloor \frac{k}{2} \rfloor + j + 1} & \cdot & g_{n-k}
\end{bmatrix}.$$

Rewriting $\mathbf{G}_{\mathcal{S}\setminus(a+j)_n}$ in block form, we get

$$\mathbf{G}_{\mathcal{S}\setminus(a+j)_n} = \left[\begin{array}{c|cc}
\mathbf{A}_j & \multicolumn{2}{c}{\mathbf{B}_j} \\
\hline
\multirow{2}{*}{$\mathbf{0}$} & \mathbf{C}_j & \mathbf{0} \\
 & \mathbf{D}_j & \mathbf{E}_j
\end{array}\right],$$

where

$$\mathbf{A}_j = \begin{bmatrix}
g_0 & g_1 & \cdot & \cdot & g_{\lceil \frac{k}{2} \rceil -1} \\
0 & g_0 & g_1 & \cdot & g_{\lceil \frac{k}{2} \rceil -2} \\
\vdots & & & & \vdots \\
0 & \cdot & & 0 & g_0
\end{bmatrix},$$

$$\mathbf{C}_j = \begin{bmatrix}
g_{n-k-1} & g_{n-k} & 0 & \cdot & \cdot & 0 \\
g_{n-k-2} & g_{n-k-1} & g_{n-k} & 0 & \cdot & 0 \\
\vdots & & & & & \vdots \\
g_{n-k-j+1} & \cdot & \cdot & \cdot & \cdot & g_{n-k} \\
g_{n-k-j} & \cdot & \cdot & \cdot & \cdot & g_{n-k-1}
\end{bmatrix},$$

and

$$\mathbf{E}_j = \begin{bmatrix}
g_{n-k} & 0 & \cdot & \cdot & 0 \\
g_{n-k-1} & g_{n-k} & 0 & \cdot & 0 \\
\vdots & & & & \vdots \\
g_{n-k-\lfloor \frac{k}{2} \rfloor + j + 1} & \cdot & \cdot & \cdot & g_{n-k}
\end{bmatrix}.$$

Matrices $\mathbf{A}_j$ and $\mathbf{E}_j$ have full rank as they are respectively upper triangular and lower triangular, with non-zero entries on the diagonal (as g_0 and g_{n-k} are non-zero in a cyclic code). Therefore, $\mathbf{G}_{\mathcal{S}\setminus(a+j)_n}$ has full rank if $\mathbf{C}_j$ has full rank. For $\lfloor\frac{k}{2}\rfloor < j < k$, $\mathbf{G}_{\mathcal{S}\setminus(a+j)_n}$ can be partitioned into a similar form and the result in Claim 4 follows.

Proof of Claim 6

We need to argue that all $k \times k$ submatrices of $\mathbf{G}_{\mathcal{S}_a}$ where $0 \le a < \alpha$ are full rank. In what follows we argue that all $k \times k$ submatrices of $\mathbf{G}_{\mathcal{S}_0}$ are full rank. The proof for any $\mathbf{G}_{\mathcal{S}_a}$ is similar. Note that $\mathbf{G}_{\mathcal{S}_0}$ can be written compactly as follows by using Kronecker products.

$$\mathbf{G}_{\mathcal{S}_0} = \begin{bmatrix} \mathbf{A} \otimes \mathbf{I}_t \\ \mathbf{B} \otimes \mathbf{C}_{(t-1)\times t}(1,1) \end{bmatrix},$$

where

$$\mathbf{A} = \begin{bmatrix} b_{00} & \cdots & b_{0(z-1)} \\ \vdots & & \vdots \\ b_{(z-2)0} & \cdots & b_{(z-2)(z-1)} \end{bmatrix}$$

and $\mathbf{B} = [b_{(z-1)0}, \cdots, b_{(z-1)(z-1)}]$.

Next, we check the determinant of submatrices $\mathbf{G}_{\mathcal{S}_0\setminus i}$ obtained by deleting i-th column of $\mathbf{G}_{\mathcal{S}_0}$. W.l.o.g, we let $i = (z-1)t + j$ where $0 \le j < t$. The block form of the resultant matrix $\mathbf{G}_{\mathcal{S}_0\setminus i}$ can be expressed as

$$\mathbf{G}_{\mathcal{S}_0\setminus i} = \begin{bmatrix} \mathbf{A}' \otimes \mathbf{I}_t & \mathbf{A}'' \otimes \Delta_1 \\ \mathbf{B}' \otimes \mathbf{C}_{(t-1)\times t}(1,1) & \mathbf{B}'' \otimes \Delta_2 \end{bmatrix},$$

where $\mathbf{A}'$ and $\mathbf{A}''$ are the first $z-1$ columns and last column of $\mathbf{A}$ respectively. Likewise, $\mathbf{B}'$ and $\mathbf{B}''$ are the first $z-1$ components and last component of $\mathbf{B}$. The matrices Δ_1 and Δ_2 are obtained by deleting the j-th column of $\mathbf{I}_t$ and $\mathbf{C}_{(t-1)\times t}(1,1)$ respectively. Then, using the Schur determinant identity Horn and Johnson (1991), we have

$$\begin{aligned}\det(\mathbf{G}_{\mathcal{S}_0\setminus i}) &= \det(\mathbf{A}'\otimes\mathbf{I}_t)\det(\mathbf{B}''\otimes\Delta_2 - \\ &\quad \mathbf{B}'\otimes\mathbf{C}_{(t-1)\times t}(1,1)\cdot(\mathbf{A}'\otimes\mathbf{I}_t)^{-1}\cdot\mathbf{A}''\otimes\Delta_1) \\ &\overset{(1)}{=} \det(\mathbf{A}'\otimes\mathbf{I}_t)\det(\mathbf{B}''\otimes\Delta_2 - \\ &\quad \mathbf{B}'\mathbf{A}'^{-1}\mathbf{A}''\otimes\mathbf{C}_{(t-1)\times t}(1,1)\Delta_1) \\ &\overset{(2)}{=} \det(\mathbf{A}'\otimes\mathbf{I}_t)\det((\mathbf{B}''-\mathbf{B}'\mathbf{A}'^{-1}\mathbf{A}'')\otimes\Delta_2),\end{aligned}$$

where (1) holds by the properties of the Kronecker product Horn and Johnson (1991) and (2) holds since $\mathbf{C}_{(t-1)\times t}(1,1)\Delta_1 = \Delta_2$. Next note that $\det(\Delta_2)\neq 0$. This is because Δ_2 can be denoted as

$$\Delta_2 = \left[\begin{array}{c|c} \mathbf{A} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{B} \end{array}\right],$$

where $\mathbf{A}$ is a $j\times j$ upper-triangular matrix;

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 1 & \cdots & 0 & 0 \\ \vdots & & & & & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

and $\mathbf{B}$ is a $(t-1-j)\times(t-1-j)$ lower-triangular matrix;

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & 0 \\ 1 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & & & & & \vdots \\ 0 & 0 & \cdots & 1 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 & 1 \end{bmatrix}.$$

Next, we define the matrix

$$\mathbf{F} = \begin{bmatrix} b_{00} & b_{01} & \cdots & b_{0(z-1)} \\ b_{10} & b_{11} & \cdots & b_{1(z-1)} \\ \vdots & & & \vdots \\ b_{(z-1)0} & b_{(z-1)1} & \cdots & b_{(z-1)(z-1)} \end{bmatrix}.$$

Another application of the Schur determinant identity yields

$$\begin{aligned} \det(\mathbf{B}'' - \mathbf{B}'\mathbf{A}'^{-1}\mathbf{A}'') &= \frac{\det(\mathbf{F})}{\det(\mathbf{A}')} \\ &\neq 0, \end{aligned}$$

since $\det(\mathbf{F})$ and $\det(\mathbf{A}')$ are both non-zero as their columns have the Vandermonde form. In $\mathbf{F}$, the columns correspond to distinct and non-zero elements from $GF(q)$; therefore, $q > z$. Note however, that the above discussion focused only $\mathbf{G}_{\mathcal{S}_0}$. As the argument needs to apply for all $\mathbf{G}_{\mathcal{S}_a}$ where $0 \leq a < \alpha$, we need $q > \alpha$.

Proof of Claim 7

Note that the matrix in eq. (3.4) is the generator matrix of (n, k) linear block code over $GF(q)$ where $nz = (z+1)(k+1)$. Since z and $z+1$ are coprime, z is the least positive integer such that $k+1 \mid nz$. To show $\mathbf{G}$ satisfies the CCP, we need to argue that all $k \times k$ submatrices of $\mathbf{G}_{\mathcal{S}_a}$ where $0 \leq a \leq z$ are full rank. It is easy to check that $\mathcal{S}_a = \{0, \cdots, n-1\} \setminus \{t(z-a), t(z-a)+1, \cdots, t(z-a)+t-1\}$. We verify three types of matrix $\mathbf{G}_{\mathcal{S}_a}$ as follows: I. $a = 0$ II. $a = 1$ III. $a > 1$.

- Type I

  When $a = 0$, it is easy to verify that any $k \times k$ submatrix of $\mathbf{G}_{\mathcal{S}_0}$ has full rank since $\mathbf{G}_{\mathcal{S}_0}$ has the form $[\mathbf{I}_{k\times k}|\mathbf{1}_k]$, which is the generator matrix of the SPC code.

- Type II

$$\mathbf{G}_{\mathcal{S}_1} = \left[\begin{array}{cc} \underbrace{\begin{matrix} \mathbf{I}_{t\times t} & \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times t} \\ \mathbf{0}_{t\times t} & \mathbf{I}_{t\times t} & \cdots & \mathbf{0}_{t\times t} \\ \vdots & & & \\ \mathbf{0}_{t\times t} & \mathbf{0}_{t\times t} & \cdots & \mathbf{I}_{t\times t} \\ \mathbf{0}_{(t-1)\times t} & \mathbf{0}_{(t-1)\times t} & \cdots & \mathbf{0}_{(t-1)\times t} \end{matrix}}_{\text{Case 1}} & \underbrace{\begin{matrix} b_1\mathbf{I}_{t\times t} \\ b_2\mathbf{I}_{t\times t} \\ \vdots \\ b_{z-1}\mathbf{I}_{t\times t} \\ \mathbf{C}(c_1,c_2)_{(t-1)\times t} \end{matrix}}_{\text{Case 2}} \end{array}\right] \tag{A.3}$$

When $a = 1$, $\mathbf{G}_{\mathcal{S}_1}$ has the form in eq. (A.3),

Case 1: Suppose that we delete any of first $(z-1)t$ columns in $\mathbf{G}_{\mathcal{S}_1}$ (this set of columns is depicted by the underbrace in eq. (A.3)), say i-th column of $\mathbf{G}_{\mathcal{S}_1}$, where $z_1 t \leq i < (z_1+1)t$ and $0 \leq z_1 \leq z-2$. Let $i_1 = i - z_1 t$, $i_2 = (z_1+1)t - i - 1$. The resultant matrix $\mathbf{G}_{\mathcal{S}_1\backslash i}$ can be expressed as follows.

$$\mathbf{G}_{\mathcal{S}_1\backslash i} = \left[\begin{array}{c|c} \mathbf{A} & \mathbf{C} \\ \hline \mathbf{B} & \mathbf{D} \end{array}\right],$$

where

$$\mathbf{A} = \mathbf{I}_{i\times i},$$

$$\mathbf{B} = \mathbf{0}_{(k-i)\times i},$$

$$\mathbf{C} = \left[\begin{array}{ccc} \mathbf{0}_{t\times(k-t-i)} & b_1\mathbf{I}_{t\times t} & \\ \mathbf{0}_{t\times(k-t-i)} & b_2\mathbf{I}_{t\times t} & \\ \vdots & \vdots & \\ \mathbf{0}_{t\times(k-t-i)} & b_{z_1}\mathbf{I}_{t\times t} & \\ \mathbf{0}_{i_1\times(k-t-i)} & b_{z_1+1}\mathbf{I}_{i_1\times i_1} & \mathbf{0}_{i_1\times(i_2+1)} \end{array}\right],$$

and $\mathbf{D}$ has the form in eq. (A.4).

Note that if $z_1 = 0$, $\mathbf{C} = [\mathbf{0}_{i_1\times(k-t-i)}\ b_1\mathbf{I}_{i_1\times i_1}\ \mathbf{0}_{i_1\times(i_2+1)}]$ and if $z_1 = z-2$,

$$\mathbf{D} = \left[\begin{array}{cccc} \mathbf{0}_{1\times i_2} & \mathbf{0}_{1\times i_1} & b_{z-1} & \mathbf{0}_{1\times i_2} \\ \mathbf{I}_{i_2\times i_2} & \mathbf{0}_{i_2\times(i_1+1)} & & b_{z-1}\mathbf{I}_{i_2\times i_2} \\ \mathbf{0}_{(t-1)\times i_2} & & \mathbf{C}(c_1,c_2)_{(t-1)\times t} & \end{array}\right].$$

$$\mathbf{D} = \begin{bmatrix} \mathbf{0}_{1\times i_2} & \mathbf{0}_{1\times t} & \cdots & \mathbf{0}_{1\times t} & \mathbf{0}_{1\times i_1} & b_{z_1+1} & \mathbf{0}_{1\times i_2} \\ \mathbf{I}_{i_2\times i_2} & \mathbf{0}_{i_2\times t} & \cdots & \mathbf{0}_{i_2\times t} & \multicolumn{2}{c}{\mathbf{0}_{i_2\times (i_1+1)}} & b_{z_1+1}\mathbf{I}_{i_2\times i_2} \\ \vdots & & & & & \vdots & \\ \mathbf{0}_{t\times i_2} & \mathbf{0}_{t\times t} & \cdots & \mathbf{I}_{t\times t} & \multicolumn{3}{c}{b_{z-1}\mathbf{I}_{t\times t}} \\ \mathbf{0}_{(t-1)\times i_2} & \mathbf{0}_{(t-1)\times t} & \cdots & \mathbf{0}_{(t-1)\times t} & \multicolumn{3}{c}{\mathbf{C}(c_1,c_2)_{(t-1)\times t}} \end{bmatrix} \tag{A.4}$$

To verify $\mathbf{G}_{\mathcal{S}_1\setminus i}$ has full rank, we just need to check $\mathbf{D}$ has full rank (as $\mathbf{A}$ is full rank). Checking that $\mathbf{D}$ has full rank can be further simplified as follows. As $b_{z_1+1} \neq 0$, we can move the corresponding column that has b_{z_1+1} as its first entry so that it is the first column of $\mathbf{D}$. Following this, consider $\mathbf{C}(c_1,c_2)_{(t-1)\times t} \setminus \mathbf{c}_{i_1}$ which is obtained by deleting the i_1-th column of $\mathbf{C}(c_1,c_2)_{(t-1)\times t}$.

$$\mathbf{C}(c_1,c_2)_{(t-1)\times t} \setminus \mathbf{c}_{i_1} = \begin{bmatrix} \mathbf{D}_1 & \mathbf{D}_2 \\ \mathbf{D}_3 & \mathbf{D}_4 \end{bmatrix},$$

where $\mathbf{D}_1$ is a $i_1 \times i_1$ matrix as follows

$$\mathbf{D}_1 = \begin{bmatrix} c_1 & c_2 & 0 & 0 & \cdots & 0 & 0 \\ 0 & c_1 & c_2 & 0 & \cdots & 0 & 0 \\ \vdots & & & & & & \vdots \\ 0 & 0 & 0 & \cdots & 0 & c_1 & c_2 \\ 0 & 0 & 0 & \cdots & 0 & 0 & c_1 \end{bmatrix},$$

$\mathbf{D}_4$ is a $i_2 \times i_2$ matrix as follows

$$\mathbf{D}_4 = \begin{bmatrix} c_2 & 0 & 0 & \cdots & 0 & 0 & 0 \\ c_1 & c_2 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & & & & & & \vdots \\ 0 & 0 & \cdots & 0 & c_1 & c_2 & 0 \\ 0 & 0 & \cdots & 0 & 0 & c_1 & c_2 \end{bmatrix},$$

and $\mathbf{D}_2$ and $\mathbf{D}_3$ are $i_1 \times i_2$ and $i_2 \times i_1$ all zero matrices respectively. Then $\det(\mathbf{C}(c_1,c_2)_{(t-1)\times t} \setminus \mathbf{c}_{i_1}) = c_1^{i_1} c_2^{i_2}$ and $\det(\mathbf{G}_{\mathcal{S}_1\setminus i}) = \pm b_{z_1+1} c_1^{i_1} c_2^{i_2} \neq 0$.

$$\mathbf{G}_{\mathcal{S}_a} = \left[\underbrace{\begin{matrix} \mathbf{I}_{t\times t} & \cdots & \mathbf{0}_{t\times t} \\ \vdots & & \vdots \\ \mathbf{0}_{t\times t} & \cdots & \mathbf{I}_{t\times t} \\ \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times t} \\ \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times t} \\ \vdots & & \vdots \\ \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times t} \\ \mathbf{0}_{(t-1)\times t} & \cdots & \mathbf{0}_{(t-1)\times t} \end{matrix}}_{\text{Case 1}} \quad \underbrace{\begin{matrix} \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times t} \\ \vdots & & \vdots \\ \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times t} \\ \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times t} \\ \mathbf{I}_{t\times t} & \cdots & \mathbf{0}_{t\times t} \\ \vdots & & \vdots \\ \mathbf{0}_{t\times t} & \cdots & \mathbf{I}_{t\times t} \\ \mathbf{0}_{(t-1)\times t} & \cdots & \mathbf{0}_{(t-1)\times(t-1)} \end{matrix}}_{\text{Case 2}} \quad \underbrace{\begin{matrix} \mathbf{0}_{t\times(t-1)} \\ \vdots \\ \mathbf{0}_{t\times(t-1)} \\ \mathbf{0}_{t\times(t-1)} \\ \mathbf{0}_{t\times(t-1)} \\ \vdots \\ \mathbf{0}_{t\times(t-1)} \\ \mathbf{I}_{(t-1)\times(t-1)} \end{matrix}}_{\text{Case 3}} \quad \underbrace{\begin{matrix} \mathbf{1}_t \\ \vdots \\ \mathbf{1}_t \\ \mathbf{1}_t \\ \mathbf{1}_t \\ \vdots \\ \mathbf{1}_t \\ \mathbf{1}_{t-1} \end{matrix}}_{\text{Case 4}} \quad \underbrace{\begin{matrix} b_1\mathbf{I}_{t\times t} \\ \vdots \\ b_{z-a}\mathbf{I}_{t\times t} \\ b_{z-a+1}\mathbf{I}_{t\times t} \\ b_{z-a+2}\mathbf{I}_{t\times t} \\ \vdots \\ b_{z-1}\mathbf{I}_{t\times t} \\ \mathbf{C}(c_1,c_2)_{(t-1)\times t} \end{matrix}}_{\text{Case 5}} \right] \tag{A.5}$$

Case 2: By deleting any of last t columns in $\mathbf{G}_{\mathcal{S}_1}$, say i-th column of $\mathbf{G}_{\mathcal{S}_1}$, where $(z-1)t \leq i < zt$, the block form of resultant matrix $\mathbf{G}_{\mathcal{S}_1\setminus i}$ can be expressed as follows.

$$\mathbf{G}_{\mathcal{S}_1\setminus i} = \left[\begin{array}{c|c} \mathbf{A} & \mathbf{C} \\ \hline \mathbf{B} & \mathbf{D} \end{array}\right],$$

where $\mathbf{A} = \mathbf{I}_{(z-1)t\times(z-1)t}$, $\mathbf{B} = \mathbf{0}_{(t-1)\times(z-1)t}$, $\mathbf{C}$ is obtained by deleting the $(i-(z-1)t)$-th column of matrix $[b_1\mathbf{I}_{t\times t}\ b_2\mathbf{I}_{t\times t}\ \cdots\ b_{z-1}\mathbf{I}_{t\times t}]^T$ and $\mathbf{D}$ is $\mathbf{C}(c_1,c_2)_{(t-1)\times t} \setminus \mathbf{c}_{i-(z-1)t}$. Since $\det(\mathbf{D}) = c_1^{i-(z-1)t}c_2^{zt-i-1} \neq 0$, $\det(\mathbf{G}_{\mathcal{S}_1\setminus i}) = \pm c_1^{i-(z-1)t}c_2^{zt-i-1} \neq 0$ and therefore $\mathbf{G}_{\mathcal{S}_1\setminus i}$ has full rank.

- Type III when $a > 1$, $\mathbf{G}_{\mathcal{S}_a}$ has the form in eq. (A.5). As before we perform a case analysis. Each of the cases is specified by the corresponding underbrace in eq. (A.5).

  Case 1: By deleting the i-th column of $\mathbf{G}_{\mathcal{S}_a}$, where $z_1 t \leq i < (z_1+1)t$, $z_1 \leq z-a-1$, $i_1 = i - z_1 t$, and $i_2 = (z_1+1)t - i - 1$, the block form of the resultant matrix $\mathbf{G}_{\mathcal{S}_a\setminus i}$ can be expressed as follows,

$$\mathbf{G}_{\mathcal{S}_a\setminus i} = \left[\begin{array}{c|c} \mathbf{A} & \mathbf{C} \\ \hline \mathbf{B} & \mathbf{D} \end{array}\right],$$

$$\mathbf{C} = \left[\begin{array}{cccccccc} \mathbf{0}_{t\times i_2} & \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times(t-1)} & \mathbf{1}_t & \multicolumn{3}{c}{b_1\mathbf{I}_{t\times t}} \\ \vdots & & & & & \multicolumn{3}{c}{\vdots} \\ \mathbf{0}_{t\times i_2} & \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times(t-1)} & \mathbf{1}_t & \multicolumn{3}{c}{b_{z_1}\mathbf{I}_{t\times t}} \\ \mathbf{0}_{i_1\times i_2} & \mathbf{0}_{i_1\times t} & \cdots & \mathbf{0}_{i_1\times(t-1)} & \mathbf{1}_{i_1} & b_{z_1+1}\mathbf{I}_{i_1\times i_1} & & \mathbf{0}_{i_1\times(i_2+1)} \end{array}\right] \tag{A.6}$$

$$\mathbf{D} = \left[\begin{array}{cccccccccc} \mathbf{0}_{1\times i_2} & \mathbf{0}_{1\times t} & \cdots & \cdots & \cdots & \mathbf{0}_{1\times(t-1)} & 1 & \mathbf{0}_{1\times i_1} & b_{z_1+1} & \mathbf{0}_{1\times i_2} \\ \mathbf{I}_{i_2\times i_2} & \mathbf{0}_{i_2\times t} & \cdots & \cdots & \cdots & \mathbf{0}_{i_2\times(t-1)} & \mathbf{1}_{i_2} & \multicolumn{2}{c}{\mathbf{0}_{i_2\times(i_1+1)}} & b_{z_1+1}\mathbf{I}_{i_2\times i_2} \\ \mathbf{0}_{t\times i_2} & \mathbf{I}_{t\times t} & \cdots & \cdots & \cdots & \mathbf{0}_{t\times(t-1)} & \mathbf{1}_t & \multicolumn{3}{c}{b_{z_1+2}\mathbf{I}_{t\times t}} \\ \vdots & & & & & \vdots & & & & \\ \mathbf{0}_{t\times i_2} & \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times(t-1)} & \mathbf{1}_t & \multicolumn{3}{c}{b_{z-a+1}\mathbf{I}_{t\times t}} \\ \mathbf{0}_{t\times i_2} & \mathbf{0}_{t\times t} & \cdots & \mathbf{I}_{t\times t} & \cdots & \mathbf{0}_{t\times(t-1)} & \mathbf{1}_t & \multicolumn{3}{c}{b_{z-a+2}\mathbf{I}_{t\times t}} \\ \vdots & & & & & & & & & \vdots \\ \mathbf{0}_{(t-1)\times i_2} & \mathbf{0}_{(t-1)\times t} & \cdots & \mathbf{0}_{(t-1)\times t} & \cdots & \mathbf{I}_{(t-1)\times(t-1)} & \mathbf{1}_{t-1} & \multicolumn{3}{c}{\mathbf{C}(c_1,c_2)_{(t-1)\times t}} \end{array}\right] \tag{A.7}$$

where $\mathbf{A} = \mathbf{I}_{i\times i}, \mathbf{B} = \mathbf{0}_{(k-i)\times i}$, $\mathbf{C}$ and $\mathbf{D}$ has the form in eq. (A.6) and eq. (A.7), respectively.

Note that if $z_1 = 0$, $\mathbf{C} = [\mathbf{0}_{i_1\times i_2}\ \mathbf{0}_{i_1\times t}\ \cdots\ \mathbf{0}_{i_1\times(t-1)}\ \mathbf{1}_{i_1}\ b_1\mathbf{I}_{i_1\times i_1}\ \mathbf{0}_{i_1\times(i_2+1)}]$ and if $z_1 = z - a - 1$, $\mathbf{D}$ has the form in (A.8).

$$\mathbf{D} = \left[\begin{array}{cccccccccc} \mathbf{0}_{1\times i_2} & \mathbf{0}_{1\times t} & \cdots & \cdots & \cdots & \mathbf{0}_{1\times(t-1)} & 1 & \mathbf{0}_{1\times i_1} & b_{z-a} & \mathbf{0}_{1\times i_2} \\ \mathbf{I}_{i_2\times i_2} & \mathbf{0}_{i_2\times t} & \cdots & \cdots & \cdots & \mathbf{0}_{i_2\times(t-1)} & \mathbf{1}_{i_2} & \multicolumn{2}{c}{\mathbf{0}_{i_2\times(i_1+1)}} & b_{z-a}\mathbf{I}_{i_2\times i_2} \\ \mathbf{0}_{t\times i_2} & \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times(t-1)} & \mathbf{1}_t & \multicolumn{3}{c}{b_{z-a+1}\mathbf{I}_{t\times t}} \\ \mathbf{0}_{t\times i_2} & \mathbf{0}_{t\times t} & \cdots & \mathbf{I}_{t\times t} & \cdots & \mathbf{0}_{t\times(t-1)} & \mathbf{1}_t & \multicolumn{3}{c}{b_{z-a+2}\mathbf{I}_{t\times t}} \\ \vdots & & & & & & & & & \vdots \\ \mathbf{0}_{(t-1)\times i_2} & \mathbf{0}_{(t-1)\times t} & \cdots & \mathbf{0}_{(t-1)\times t} & \cdots & \mathbf{I}_{(t-1)\times(t-1)} & \mathbf{1}_{t-1} & \multicolumn{3}{c}{\mathbf{C}(c_1,c_2)_{(t-1)\times t}} \end{array}\right]. \tag{A.8}$$

To verify that $\mathbf{G}_{\mathcal{S}_a\setminus i}$ has full rank, we just need to check $\mathbf{D}$ has full rank. Owing to the construction of $\mathbf{D}$, we have to check the determinant of the following $(t+1)\times(t+1)$ matrix.

$$\mathbf{F}=\begin{bmatrix} 1 & 0 & 0 & \cdots & b_{z_1+1} & \cdots & 0 \\ 1 & b_{z-a+1} & 0 & \cdots & 0 & \cdots & 0 \\ 1 & 0 & b_{z-a+1} & \cdots & 0 & \cdots & 0 \\ \vdots & & & & \vdots & & \\ 1 & 0 & 0 & \cdots & 0 & \cdots & b_{z-a+1} \end{bmatrix};$$

$\det(\mathbf{F})=(b_{z-a+1}-b_{z_1+1})b_{z-a+1}^{t-1}$. Since $z_1\neq z-a$ and then $b_{z_1+1}\neq b_{z-a+1}$, the above matrix has full rank and $\det(\mathbf{G}_{\mathcal{S}_a\setminus i})=\pm(b_{z-a+1}-b_{z_1+1})b_{z-a+1}^{t-1}\neq 0$, so that $\mathbf{G}_{\mathcal{S}_a\setminus i}$ has full rank.

Case 2: By deleting i-th column of $\mathbf{G}_{\mathcal{S}_a}$, where $z_1t\leq i<(z_1+1)t$, $z-a\leq z_1\leq z-3$, the proof that the resultant matrix has full rank is similar to the case that $z_1\leq z-a-1$ and we omit it here.

Case 3: By deleting i-th column of $\mathbf{G}_{\mathcal{S}_a}$, where $(z-2)t\leq i\leq(z-1)t-2$, $i_1=i-(z-2)t$ and $i_2=(z-1)t-2-i$, the resultant matrix is as follows,

$$\mathbf{G}_{\mathcal{S}_a\setminus i}=\left[\begin{array}{c|c} \mathbf{A} & \mathbf{C} \\ \hline \mathbf{B} & \mathbf{D} \end{array}\right],$$

where

$$\mathbf{A}=\mathbf{I}_{(z-a)t\times(z-a)t}$$

$$\mathbf{B}=\mathbf{0}_{(k-(z-a)t)\times(z-a)t}$$

$$\mathbf{C}=\begin{bmatrix} \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times(t-2)} & \mathbf{1}_t & b_1\mathbf{I}_{t\times t} \\ \vdots & & & & \vdots \\ \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times(t-2)} & \mathbf{1}_t & b_{z-a}\mathbf{I}_{t\times t} \end{bmatrix}$$

$$\mathbf{D}=\begin{bmatrix} \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times(t-2)} & & \mathbf{1}_t & b_{z-a+1}\mathbf{I}_{t\times t} \\ \mathbf{I}_{t\times t} & \cdots & \mathbf{0}_{t\times(t-2)} & & \mathbf{1}_t & b_{z-a+2}\mathbf{I}_{t\times t} \\ \vdots & & & & \vdots & \\ & & \mathbf{I}_{i_1\times i_1} & \mathbf{0}_{i_1\times i_2} & & \\ \mathbf{0}_{(t-1)\times t} & \cdots & \mathbf{0}_{1\times i_1} & \mathbf{0}_{1\times i_2} & \mathbf{1}_{t-1} & \mathbf{C}(c_1,c_2)_{(t-1)\times t} \\ & & \mathbf{0}_{i_2\times i_1} & \mathbf{I}_{i_2\times i_2} & & \end{bmatrix}$$

$$
\mathbf{G}_{\mathcal{S}_a\setminus i} = \begin{bmatrix} \mathbf{I}_{t\times t} & \cdots & \mathbf{0}_{t\times t} & \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times(t-1)} & b_1\mathbf{I}_{t\times t} \\ \vdots & & & & & & \vdots \\ \mathbf{0}_{t\times t} & \cdots & \mathbf{I}_{t\times t} & \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times(t-1)} & b_{z-a}\mathbf{I}_{t\times t} \\ \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times t} & \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times(t-1)} & b_{z-a+1}\mathbf{I}_{t\times t} \\ \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times t} & \mathbf{I}_{t\times t} & \cdots & \mathbf{0}_{t\times(t-1)} & b_{z-a+2}\mathbf{I}_{t\times t} \\ \vdots & & & & & & \vdots \\ \mathbf{0}_{(t-1)\times t} & \cdots & \mathbf{0}_{(t-1)\times t} & \mathbf{0}_{(t-1)\times t} & \cdots & \mathbf{I}_{(t-1)\times(t-1)} & \mathbf{C}(c_1,c_2)_{(t-1)\times t} \end{bmatrix} \tag{A.9}
$$

To verify $\mathbf{G}_{\mathcal{S}_a\setminus i}$ has full rank, we need to check the determinant of $\mathbf{D}$. Owing to the construction of $\mathbf{D}$, the following matrix is required to be full rank,

$$
\begin{aligned}
\mathbf{D}' &= \begin{bmatrix} \mathbf{1}_t & b_{z-a+1}\mathbf{I}_{t\times t} \\ & \\ 1 & \mathbf{C}(c_1,c_2)_{(t-1)\times t}(i_1) \end{bmatrix} \\
&= \begin{bmatrix} 1 & b_{z-a+1} & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 1 & 0 & b_{z-a+1} & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & & & & & & & & \vdots \\ 1 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & b_{z-a+1} \\ 1 & 0 & \cdots & 0 & c_1 & c_2 & 0 & \cdots & 0 \end{bmatrix},
\end{aligned}
$$

where $\mathbf{C}(c_1,c_2)_{(t-1)\times t}(i_1)$ denotes the i_1-th row of $\mathbf{C}(c_1,c_2)_{(t-1)\times t}$, $0 \le i \le t-2$.

$$
\begin{aligned}
\det\mathbf{D}' =& \det(b_{z-a+1}\mathbf{I}_{t\times t}) \cdot \det(1- \\
& \mathbf{C}(c_1,c_2)_{(t-1)\times t}(i_1) \cdot (b_{z-a+1}^{-1}\mathbf{I}_{t\times t}) \cdot \mathbf{1}_t) \\
=& b_{z-a+1}^{t}(1 - b_{z-a+1}^{-1}(c_1+c_2))
\end{aligned}
$$

Since $b_{z-a+1} \neq 0$ and $c_1 + c_2 = 0$, $\det\mathbf{D}' \neq 0$ and $\mathbf{D}'$ has full rank. Then $\det(\mathbf{D}) = b_{z-a+1}^{t}(1 - b_{z-a+1}^{-1}(c_1+c_2)) \neq 0$ and thus $\mathbf{G}_{\mathcal{S}_a\setminus i}$ is full rank.

Case 4: By deleting i-th column of $\mathbf{G}_{\mathcal{S}_a}$, where $i = (z-1)t-1$, the block form of the resultant matrix $\mathbf{G}_{\mathcal{S}_a\setminus i}$ can be expressed as eq. (A.9). Evidently, $\det(\mathbf{G}_{\mathcal{S}_a\setminus i}) = \pm b_{z-a+1}^{t}$, so that $\mathbf{G}_{\mathcal{S}_a\setminus i}$ has full rank.

Case 5: By deleting i-th column of $\mathbf{G}_{\mathcal{S}_a}$, where $(z-1)t \le i < zt$ and $i_1 = i - (z-1)t$, the block form of the resultant matrix $\mathbf{G}_{\mathcal{S}_a\setminus i}$ can be expressed as eq. (A.10).

where $b_s\mathbf{I}_{t\times t} \setminus \mathbf{c}_{i_1}$ denotes the submatrix obtained by deleting i_1-th column of $b_s\mathbf{I}_{t\times t}$ and $\mathbf{C}(c_1,c_2)_{(t-1)\times t}\setminus\mathbf{c}_{i_1}$ denotes the submatrix obtained by deleting i_1-th column of $\mathbf{C}(c_1,c_2)_{(t-1)\times t}\setminus$

$$\mathbf{G}_{\mathcal{S}_a\setminus i} = \begin{bmatrix} \mathbf{I}_{t\times t} & \cdots & \mathbf{0}_{t\times t} & \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times (t-1)} & \mathbf{1}_t & b_1\mathbf{I}_{t\times t}\setminus \mathbf{c}_{i_1} \\ \vdots & & & & & & & \vdots \\ \mathbf{0}_{t\times t} & \cdots & \mathbf{I}_{t\times t} & \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times (t-1)} & \mathbf{1}_t & b_{z-a}\mathbf{I}_{t\times t}\setminus \mathbf{c}_{i_1} \\ \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times t} & \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times (t-1)} & \mathbf{1}_t & b_{z-a+1}\mathbf{I}_{t\times t}\setminus \mathbf{c}_{i_1} \\ \mathbf{0}_{t\times t} & \cdots & \mathbf{0}_{t\times t} & \mathbf{I}_{t\times t} & \cdots & \mathbf{0}_{t\times (t-1)} & \mathbf{1}_t & b_{z-a+2}\mathbf{I}_{t\times t}\setminus \mathbf{c}_{i_1} \\ \vdots & & & & & & & \vdots \\ \mathbf{0}_{(t-1)\times t} & \cdots & \mathbf{0}_{(t-1)\times t} & \mathbf{0}_{(t-1)\times t} & \cdots & \mathbf{I}_{(t-1)\times (t-1)} & \mathbf{1}_{t-1} & \mathbf{C}(c_1,c_2)_{(t-1)\times t}\setminus \mathbf{c}_{i_1} \end{bmatrix} \tag{A.10}$$

$\mathbf{c}_{i_1}$. To verify $\mathbf{G}_{\mathcal{S}_a\setminus i}$ has full rank, we just need to check $[\mathbf{1}_t|b_{z-a+1}\mathbf{I}_{t\times t}\setminus \mathbf{c}_{i_1}]$ has full rank. Since $[\mathbf{1}_t|b_{z-a+1}\mathbf{I}_{t\times t}]$ has the following form,

$$\begin{bmatrix} 1 & b_{z-a+1} & 0 & \cdots & 0 \\ 1 & 0 & b_{z-a+1} & \cdots & 0 \\ \vdots & & & & \vdots \\ 1 & 0 & 0 & \cdots & b_{z-a+1} \end{bmatrix},$$

by deleting any column of above matrix, it is obvious that $\det([\mathbf{1}_t|b_{z-a+1}\mathbf{I}_{t\times t}\setminus \mathbf{c}_{i_1}]) = \pm b_{z-a+1}^{t-1}$ and $\det(\mathbf{G}_{\mathcal{S}_a\setminus i}) \neq 0$.

Proof of Claim 8

Let z be the least integer such that $k+1 \mid nz$. First, we argue that z is the least integer such that $k+1 \mid (n+s(k+1))z$. Assume that this is not true, then there exists $z' < z$ such that $k+1 \mid (n+s(k+1))z'$. As $n \geq k+1$ and $k+1 \mid s(k+1)z'$ this implies that $k+1 \mid nz'$ which is a contradiction.

Next we argue that $\mathbf{G}'$ satisfies the CCP, i.e., all $k \times k$ submatrices of each $\mathbf{G}'_{\mathcal{S}'_a}$, where $\mathcal{T}'_a = \{a(k+1), \cdots, a(k+1)+k\}$ and $\mathcal{S}'_a = \{(t)_{n+s(k+1)} | t \in \mathcal{T}'_a\}$ and $0 \leq a \leq \frac{nz}{k+1} + sz$, are full rank. Let $n' = n + s(k+1)$. We argue it in three cases.

- *Case 1. The first column of $\mathbf{G}'_{\mathcal{S}'_a}$ lies in the first $s(k+1)$ columns of $\mathbf{G}'$.*

  Suppose $ln' \leq a(k+1) < ln' + s(k+1)$ where $0 \leq l < z-1$. By the construction of $\mathbf{G}'$, $\mathbf{G}'_{\mathcal{S}_a} = \mathbf{D}$. Since $\mathbf{D} = \mathbf{G}_{\mathcal{S}_0}$, all $k \times k$ submatrices of $\mathbf{D}$ have full rank and so does $\mathbf{G}'_{\mathcal{S}_a}$.

- *Case 2. The first and last column of $\mathbf{G}'_{\mathcal{S}'_a}$ lie in the last n columns of $\mathbf{G}'$.*

  Suppose $ln' + s(k+1) \leq a(k+1)$ and $a(k+1) + k < (l+1)n'$ where $0 \leq l < z-1$. As $n' > s(k+1)$, $a(k+1) - (l+1)s(k+1) > 0$ and $k+1|a(k+1) - (l+1)s(k+1)$. Let $a' = a - (l+1)s$, then $\mathbf{G}'_{\mathcal{S}_a} = \mathbf{G}_{\mathcal{S}_{a'}}$ and hence all $k \times k$ submatrices of $\mathbf{G}'_{\mathcal{S}_a}$ have full rank.

- *Case 3. The first column of $\mathbf{G}'_{\mathcal{S}'_a}$ lies in the last n columns of $\mathbf{G}'$ but the last column lies in the first $(k+1)$ columns of $\mathbf{G}'$.*

  Suppose $ln' + s(k+1) \leq a(k+1)$ and $a(k+1) + k > (l+1)n'$ where $0 \leq l < z-2$. Again, we can get $k+1|a(k+1) - (l+1)s(k+1)$ and let $a' = a - (l+1)s$. Let $\mathcal{S}'^1 = \{(a(k+1))_{n'}, \cdots, (ln' + n' - 1)_{n'}\}$ and $\mathcal{S}^1 = \{(a'(k+1))_n, \cdots, (ln + n - 1)_n\}$. As $(ln' + n' - 1) - a(k+1) = (ln + n - 1) - a'(k+1)$, $\mathbf{G}'_{\mathcal{S}'^1} = \mathbf{G}_{\mathcal{S}^1}$. Let $\mathcal{S}'^2 = \{(ln' + n')_{n'}, \cdots, (a(k+1) + k)_{n'}\}$ and $\mathcal{S}^2 = \{(ln + n)_n, \cdots, (a'(k+1) + k)_n\}$. By the construction of $\mathbf{G}'$, $\mathbf{G}_{\mathcal{S}^2} = \mathbf{G}'_{\mathcal{S}'^2}$. Then $\mathbf{G}'_{\mathcal{S}_a} = [\mathbf{G}'_{\mathcal{S}'^1}\mathbf{G}'_{\mathcal{S}'^2}] = [\mathbf{G}_{\mathcal{S}^1}\mathbf{G}_{\mathcal{S}^2}] = \mathbf{G}_{\mathcal{S}_{a'}}$ and hence all $k \times k$ submatrices of $\mathbf{G}'_{\mathcal{S}_a}$ have full rank.

Proof of Claim 11

- $\frac{M}{N} = \frac{1}{q}$. We have

$$\frac{1}{K}\log_2 \frac{F_s^{MN}}{F_s^*} = \frac{1}{K}\log_2 \binom{K}{K/q} - \frac{1}{K}\log_2 z - \frac{k}{K}\log_2 q.$$

  Using the fact that $z \leq k+1$ and taking limits as $n \to \infty$, we get that

$$\lim_{n\to\infty} \frac{1}{K}\log_2 \frac{F_s^{MN}}{F_s^*} = H_2\left(\frac{1}{q}\right) - \frac{\eta}{q}\log_2 q.$$

- $\frac{M}{N} = 1 - \frac{k+1}{nq}$. We have

$$\begin{aligned}\frac{1}{K}\log_2 \frac{F_s^{MN}}{F_s^*} =& \frac{1}{K}\log_2 \binom{K}{k+1} - \frac{k+1}{K}\log_2 q \\ &- \frac{1}{K}\log_2 \frac{zn}{k+1}\end{aligned}$$

Using the fact that $z \leq k+1$ and taking limits as $n \to \infty$, we get that

$$\lim_{n\to\infty} \frac{1}{K} \log_2 \frac{F_s^{MN}}{F_s^*} = H_2\left(\frac{\eta}{q}\right) - \frac{\eta}{q} \log_2 q$$

Discussion on coded caching systems constructed by generator matrices satisfying the (k, α)-CCP where $\alpha \leq k$

Consider the (k, α)-CCP (*cf.* Definition 6) where $\alpha \leq k$. Let z be the least integer such that $\alpha \mid nz$, and let $\mathcal{T}_a^\alpha = \{a\alpha, \cdots, a\alpha + \alpha - 1)\}$ and $\mathcal{S}_a^\alpha = \{(t)_n \mid t \in \mathcal{T}_a^\alpha\}$. Let $\mathbf{G}_{\mathcal{S}_a^\alpha} = [\mathbf{g}_{i_0}, \cdots, \mathbf{g}_{i_{\alpha-1}}]$ be the submatrix of $\mathbf{G}$ specified by the columns in $\mathcal{S}_a^\alpha$, i.e, $\mathbf{g}_{i_j} \in \mathbf{G}_{\mathcal{S}_a^\alpha}$ if $i_j \in \mathcal{S}_a^\alpha$. We demonstrate that the resolvable design generated from a linear block code that satisfies the (k, α)-CCP can also be used in a coded caching scheme. First, we construct a $(X, \mathcal{A})$ resolvable design as described in Section 3.2.A., which can be partitioned into n parallel classes $\mathcal{P}_i = \{B_{i,j} : 0 \leq j < q\}$, $0 \leq i < n$. By the constructed resolvable design, we partition each subfile W_n into $q^k z$ subfiles $W_n = \{W_{n,t}^s \mid 0 \leq t < q^k, 0 \leq s < z\}$ and operate the placement scheme in Algorithm 2. In the delivery phase, for each recovery set, several equations are generated, each of which benefit α users simultaneously. Furthermore, the equations generated by all the recovery sets can recover all the missing subfiles. In this section, we only show that for the recovery set $\mathcal{P}_{\mathcal{S}_a^\alpha}$, it is possible to generate equations which benefit α users and allow the recovery of all of missing subfiles with given superscript. The subsequent discussion exactly mirrors the discussion in the $(k, k+1)$-CCP case and is skipped.

Towards this end, we first show that picking α users from α distinct parallel classes can always form $q^{k-\alpha+1} - q^{k-\alpha}$ signals. More specifically, consider blocks $B_{i_1,l_{i_1}}, \ldots, B_{i_\alpha,l_{i_\alpha}}$ (where $l_{i_j} \in \{0, \ldots, q-1\}$) that are picked from α distinct parallel classes of $\mathcal{P}_{\mathcal{S}_a^\alpha}$. Then, $|\cap_{j=1}^{\alpha-1} B_{i_j,l_{i_j}}| = q^{k-\alpha+1}$ and $|\cap_{j=1}^{\alpha} B_{i_j,l_{i_j}}| = q^{k-\alpha}$.

Claim 13. Consider the resolvable design $(X, \mathcal{A})$ constructed by a (n, k) linear block code that satisfies the (k, α) CCP. Let $\mathcal{P}_{\mathcal{S}_a^\alpha} = \{\mathcal{P}_i \mid i \in \mathcal{S}_a^\alpha\}$ for $0 \leq a < \frac{zn}{\alpha}$, i.e., it is the set of parallel classes corresponding to $\mathcal{S}_a^\alpha$. We emphasize that $|\mathcal{P}_{\mathcal{S}_a^\alpha}| = \alpha \leq k$. Consider blocks $B_{i_1,l_{i_1}}, \ldots, B_{i_{\alpha'},l_{i_{\alpha'}}}$ (where

$l_{i_j} \in \{0, \dots, q-1\}$) that are picked from any α' distinct parallel classes of $\mathcal{P}_{\mathcal{S}_a^\alpha}$ where $\alpha' \leq \alpha$. Then, $| \cap_{j=1}^{\alpha'} B_{i_j, l_{i_j}} | = q^{k-\alpha'}$.

The above argument implies that any $\alpha - 1$ blocks from any $\alpha - 1$ distinct parallel classes of $\mathcal{P}_{\mathcal{S}_a^\alpha}$ have $q^{k-\alpha+1}$ points in common and any α blocks $B_{i_1, l_{i_1}}$, $B_{i_\alpha, l_{i_\alpha}}$ from any α distinct parallel classes of $\mathcal{P}_{\mathcal{S}_a^\alpha}$ have $q^{k-\alpha}$ points in common. These blocks (or users) can participate in $q^{k-\alpha+1} - q^{k-\alpha}$ equations, each of which benefits α users. In particular, each user will recover a missing subfile indexed by an element belonging to the intersection of the other $\alpha - 1$ blocks in each equation. A very similar argument to Lemma 2 can be made to justify enough equations can be found that allow all users to recover all missing subfiles.

Proof. Recall that by the construction in Section III.A, block $B_{i,l} \in \mathcal{P}_i$ is specified as follows,

$$B_{i,l} = \{j : \mathbf{T}_{i,j} = l\}.$$

Let $\mathbf{G} = [g_{ab}]$, for $0 \leq a < k$, $0 \leq b < n$.

Now consider $B_{i_1, l_{i_1}}, \dots, B_{i_{\alpha'}, l_{i_{\alpha'}}}$ (where $i_j \in \mathcal{S}_a^\alpha, l_{i_j} \in \{0, \dots, q-1\}$) that are picked from α' distinct parallel classes of $\mathcal{P}_{\mathcal{S}_a^\alpha}$. W.l.o.g. we assume that $i_1 < i_2 < \cdots < i_{\alpha'}$. Let $\mathcal{I} = \{i_1, \dots, i_{\alpha'}\}$ and $\mathbf{T}_\mathcal{I}$ denote the submatrix of $\mathbf{T}$ obtained by retaining the rows in $\mathcal{I}$. We will show that the vector $[l_{i_1} \ l_{i_2} \ \dots \ l_{i_{\alpha'}}]^T$ is a column in $\mathbf{T}_\mathcal{I}$ and appears $q^{k-\alpha'}$ times in it.

We note here that by the (k, α)-CCP, the vectors $\mathbf{g}_{i_1}, \mathbf{g}_{i_2}, \dots, \mathbf{g}_{i_\alpha}$ are linearly independent and thus the subset of these vectors, $\mathbf{g}_{i_1}, \cdots, \mathbf{g}_{i_{\alpha'}}$ are linearly independent. W. l. o. g., we assume that the top $\alpha' \times \alpha'$ submatrix of the matrix $[\mathbf{g}_{i_1} \ \mathbf{g}_{i_2} \ \dots \ \mathbf{g}_{i_{\alpha'}}]$ is full-rank. Next, consider the system of equations in variables $\mathbf{u}_0, \dots, \mathbf{u}_{\alpha'-1}$.

$$\sum_{b=0}^{\alpha'-1} \mathbf{u}_b g_{bi_1} = l_{i_1} - \sum_{b=\alpha'}^{k-1} \mathbf{u}_b g_{bi_1},$$
$$\sum_{b=0}^{\alpha'-1} \mathbf{u}_b g_{bi_2} = l_{i_2} - \sum_{b=\alpha'}^{k-1} \mathbf{u}_b g_{bi_2},$$
$$\vdots$$
$$\sum_{b=0}^{\alpha'-1} \mathbf{u}_b g_{bi_{\alpha'}} = l_{i_{\alpha'}} - \sum_{b=\alpha'}^{k-1} \mathbf{u}_b g_{bi_{\alpha'}}.$$

By the assumed condition, it is evident that this system of α' equations in α' variables has a unique solution for a given vector $\mathbf{v} = [\mathbf{u}_{\alpha'}, \cdots, \mathbf{u}_{k-1}]$ over $GF(q)$. Since there are $q^{k-\alpha'}$ possible $\mathbf{v}$ vectors, the result follows. □

As in the case of the $(k, k+1)$-CCP, we form a recovery set bipartite graph with parallel classes and recovery sets as the disjoint vertex subsets, and the edges incident on each parallel class are labeled arbitrarily from 0 to $z-1$. For a parallel class $\mathcal{P} \in \mathcal{P}_{\mathcal{S}_a^\alpha}$ we denote this label by label$(\mathcal{P} - \mathcal{P}_{\mathcal{S}_a^\alpha})$. For a given recovery set $\mathcal{P}_{\mathcal{S}_a^\alpha}$, the delivery phase proceeds by choosing blocks from α distinct parallel classes in $\mathcal{P}_{\mathcal{S}_a^\alpha}$ and it provides $q^{k-\alpha+1} - q^{k-\alpha}$ equations that benefit α users. Note that in the (k, α)-CCP case, randomly picking α blocks from α parallel classes in $\mathcal{P}_{\mathcal{S}_a^\alpha}$ will always result in $q^{k-\alpha}$ intersections, which is different from $(k, k+1)$-CCP. It turns out that each equation allows a user in $\mathcal{P} \in \mathcal{P}_{\mathcal{S}_a^\alpha}$ to recover a missing subfile with superscript label$(\mathcal{P} - \mathcal{P}_{\mathcal{S}_a^\alpha})$.

Let the demand of user $U_{B_{i,j}}$ for $i \in n-1, 0 \leq j \leq q-1$ by $W_{\kappa_{i,j}}$. We formalize the argument in Algorithm 6 and prove that equations generated in each recovery set $\mathcal{P}_{\mathcal{S}_a^\alpha}$ can recover all missing subfile with superscript label$(\mathcal{P} - \mathcal{P}_{\mathcal{S}_a^\alpha})$.

Algorithm 6: Signal Generation Algorithm for $\mathcal{P}_{\mathcal{S}_a^\alpha}$

```
Input  : For $\mathcal{P} \in \mathcal{P}_{\mathcal{S}_a^\alpha}$, $E(\mathcal{P}) = $ label$(\mathcal{P} - \mathcal{P}_{\mathcal{S}_a^\alpha})$. Signal set $Sig = \emptyset$.
while any user $U_B \in \mathcal{P}_j, j \in \mathcal{S}_a^\alpha$ does not recover all its missing subfiles with superscript $E(\mathcal{P})$ do
    Pick blocks $B_{j,l_j} \in \mathcal{P}_j$ for all $j \in \mathcal{S}_a^\alpha$ and $l_j \in \{0, \ldots, q-1\}$;
    /* Pick blocks from distinct parallel classes in $\mathcal{P}_{\mathcal{S}_a^\alpha}$. The cardinality of their
       intersection is always $q^{k-\alpha}$ */
    Find set $\hat{L}_s = \cap_{j \in \mathcal{S}_a^\alpha \setminus \{s\}} B_{j,l_j} \setminus \cap_{j \in \mathcal{S}_a^\alpha} B_{j,l_j}$ for $s \in \mathcal{S}_a^\alpha$;
    /* Determine the missing subfile indices that the user from $\mathcal{P}_s^\alpha$ will recover. Note that
       $|\hat{L}_s| = q^{k-\alpha+1} - q^{k-\alpha}$ */
    Add signals $\oplus_{s \in \mathcal{S}_a^\alpha} W^{E(\mathcal{P}_s)}_{\kappa_{s,l_s}, \hat{L}_s[t]}$, $0 \leq t < q^{k-\alpha+1} - q^{k-\alpha}$, to $Sig$;
    /* User $U_{B_{s,l_s}}$ demands file $W_{\kappa_{s,l_s}}$. This equation allows it to recover the corresponding
       missing subfile index $\hat{L}_s[t]$, which is the $t$-th element of $\hat{L}_s[t]$. The superscript is
       determined by the recovery set bipartite graph */
end
Output: Signal set $Sig$.
```

For the sake of convenience we argue that user $U_{B_{\beta,l_\beta}}$ that demands $W_{\kappa_{\beta,l_\beta}}$ can recover all its missing subfiles with superscript $E(\mathcal{P}_\beta)$. Note that $B_{\beta,l_\beta} = q^{k-1}$. Thus user $U_{B_{\beta,l_\beta}}$ needs to obtain $q^k - q^{k-1}$ missing subfiles with superscript $E(\mathcal{P}_\beta)$. The delivery phase scheme repeatedly picks α users from different parallel classes of $\mathcal{P}_{\mathcal{S}_a^\alpha}$. The equations in Algorithm 6 allow $U_{B_{\beta,l_\beta}}$ to recover all $W^{E(\mathcal{P}_\beta)}_{\kappa_{\beta,l_\beta},\hat{L}_\beta[t]}$ where $\hat{L}_\beta = \cap_{j\in\mathcal{S}_a^\alpha\setminus\{\beta\}} B_{j,l_j} \setminus \cap_{j\in\mathcal{S}_a^\alpha} B_{j,l_j}$ and $t = 1, \cdots, q^{k-\alpha+1} - q^{k-\alpha}$. This is because of Claim 13.

Next, we count the number of equations that $U_{B_{\beta,l_\beta}}$ participates in. We can pick $\alpha - 1$ users from $\alpha - 1$ parallel classes in $\mathcal{P}_{\mathcal{S}_a^\alpha}$. There are totally $q^{\alpha-1}$ ways to pick them, each of which generate $q^{k-\alpha+1} - q^{k-\alpha}$ equations. Thus there are a total of $q^k - q^{k-1}$ equations in which user $U_{B_{\beta,l_\beta}}$ participates in.

It remains to argue that each equation provides a distinct file part of user $U_{B_{\beta,l_\beta}}$. Towards this end, let $\{i_1, \cdots, i_{\alpha-1}\} \subset \mathcal{S}_a^\alpha$ be an index set such that $\beta \notin \{i_1, \cdots, i_{\alpha-1}\}$ but $\beta \in \mathcal{S}_a^\alpha$. Note that when we pick the same set of blocks $\{B_{i_1,l_{i_1}}, \cdots, B_{i_{\alpha-1},l_{i_{\alpha-1}}}\}$, it is impossible that the recovered subfiles $W^{E(\mathcal{P}_\beta)}_{\kappa_{\beta,l_\beta},\hat{L}_\beta[t_1]}$ and $W^{E(\mathcal{P}_\beta)}_{\kappa_{\beta,l_\beta},\hat{L}_\beta[t_2]}$ are the same since the points in $\hat{L}_\beta$ are distinct. Next, suppose that there exist sets of blocks $\{B_{i_1,l_{i_1}}, \cdots, B_{i_{\alpha-1},l_{i_{\alpha-1}}}\}$ and $\{B_{i_1,l'_{i_1}}, \cdots, B_{i_{\alpha-1},l'_{i_{\alpha-1}}}\}$ such that $\{B_{i_1,l_{i_1}}, \cdots, B_{i_{\alpha-1},l_{i_{\alpha-1}}}\} \neq \{B_{i_1,l'_{i_1}}, \cdots, B_{i_{\alpha-1},l'_{i_{\alpha-1}}}\}$, but $\gamma \in \cap_{j=1}^{\alpha-1} B_{i_j,l_{i_j}} \setminus B_{\beta,l_\beta}$ and $\gamma \in \cap_{j=1}^{\alpha-1} B_{i_j,l'_{i_j}} \setminus B_{\beta,l'_\beta}$. This is a contradiction since this in turn implies that $\gamma \in \cap_{j=2}^{\alpha} B_{i_j,l_{i_j}} \bigcap \cap_{j=2}^{\alpha} B_{i_j,l'_{i_j}}$, which is impossible since two blocks from the same parallel class have an empty intersection.

Finally we calculate the transmission rate. In Algorithm 6, for each recovery set, we transmit $q^{k+1} - q^k$ equations and there are totally $\frac{zn}{\alpha}$ recovery sets. Since each equation has size equal to a subfile, the rate is given by

$$\begin{aligned} R &= (q^{k+1} - q^k) \times \frac{zn}{\alpha} \times \frac{1}{zq^k} \\ &= \frac{n(q-1)}{\alpha}. \end{aligned}$$

The (n, k) linear block codes that satisfy the (k, α)-CCP over $GF(q)$ correspond to a coded caching system with $K = nq$, $\frac{M}{N} = \frac{1}{q}$, $F_s = zq^k$ and have a rate $R = \frac{n(q-1)}{\alpha}$. Thus, the rate of this system is a little higher compared to the $(k, k+1)$-CCP system with almost the same subpacketization level.

However, by comparing Definitions 5 and 6 it is evident that the rank constraints of the (k, α)-CCP are weaker as compared to the $(k, k+1)$-CCP. Therefore, in general we can find more instances of generator matrices that satisfy the (k, α)-CCP. For example, a large class of codes that satisfy the (k, k)-CCP are (n, k) cyclic codes since any k consecutive columns in their generator matrices are linearly independent Lin and Costello (2004). Thus, (n, k) cyclic codes always satisfy the (k, k)-CCP but satisfy $(k, k+1)$-CCP if they satisfy the additional constraints discussed in Claim 3.

Cyclic codes over $\mathbb{Z} \mod q$ Blake (1972)

First, we show that matrix $\mathbf{T}$ constructed by constructed by the approach outlined in Section 3.3.4 still results in a resolvable design. Let $\Delta = [\Delta_0 \Delta_1 \cdots \Delta_{n-1}]$ be a codeword of the cyclic code over $\mathbb{Z} \mod q$, denoted $\mathcal{C}$ where $q = q_1 q_2 \cdots q_d$, and $q_i, i = 1, \ldots, d$ are prime. By using the Chinese remaindering map ψ (discussed in Section 3.3.4), Δ can be uniquely mapped into d codewords $\mathbf{c}^{(i)}, i = 1, \ldots, d$ where each $\mathbf{c}^{(i)}$ is a codeword of $\mathcal{C}^i$ (the cyclic code over $GF(q_i)$). Thus, the b-th component Δ_b can be mapped to $(\mathbf{c}_b^{(1)}, \mathbf{c}_b^{(2)}, \ldots, \mathbf{c}_b^{(d)})$

Let $\mathbf{G}^i = [g_{ab}^{(i)}]$ represent the generator matrix of the code $\mathcal{C}^i$. Based on prior arguments, it is evident that there are $q_i^{k_i - 1}$ distinct solutions over $GF(q_i)$ to the equation $\sum_{a=0}^{k_i-1} \mathbf{u}_a g_{ab}^{(i)} = \mathbf{c}_b^{(i)}$. In turn, this implies that Δ_b appears $q_1^{k_1-1} q_2^{k_2-1} \cdots q_d^{k_d-1}$ times in the b-th row of $\mathbf{T}$ and the result follows.

Next we show any α blocks from distinct parallel classes of $\mathcal{P}_{\mathcal{S}_a^{k_{min}}}$ have $q_1^{k_1-\alpha} q_2^{k_2-\alpha} \cdots q_d^{k_d-\alpha}$ intersections, where $\alpha \leq k_{min}$ and $\mathcal{S}_a^{k_{min}} = \{(ak_{min})_n, (ak_{min}+1)_n, \cdots, (ak_{min}+k_{min}-1)_n\}$

Towards this end consider $B_{i_1, l_{i_1}}, \ldots, B_{i_\alpha, l_{i_\alpha}}$ (where $i_j \in \mathcal{S}_a^{k_{min}}, l_{i_j} \in \{0, \ldots, q-1\}$) that are picked from α distinct parallel classes of $\mathcal{P}_{\mathcal{S}_a^{k_{min}}}$. W.l.o.g. we assume that $i_1 < i_2 < \cdots < i_\alpha$. Let $\mathcal{I} = \{i_1, \ldots, i_\alpha\}$ and $\mathbf{T}_\mathcal{I}$ denote the submatrix of $\mathbf{T}$ obtained by retaining the rows in $\mathcal{I}$. We will show that the vector $[l_{i_1} \ l_{i_2} \ \ldots \ l_{i_\alpha}]^T$ is a column in $\mathbf{T}_\mathcal{I}$ and appears $q_1^{k_1-\alpha} q_2^{k_2-\alpha} \cdots q_d^{k_d-\alpha}$ times.

Let $\psi_m(l_{i_j})$ for $m = 1, \ldots, d$ represent the m-th component of the map ψ. Consider the (n, k_1) cyclic code over $GF(q_1)$ and the system of equations in variables $\mathbf{u}_0, \ldots, \mathbf{u}_{\alpha-1}$ that lie in $GF(q_1)$.

$$\begin{aligned}
\sum_{b=0}^{\alpha-1} \mathbf{u}_b g_{bi_1}^{(1)} &= \psi_1(l_{i_1}) - \sum_{b=\alpha}^{k_1-1} \mathbf{u}_b g_{bi_1}^{(1)}, \\
\sum_{b=0}^{\alpha-1} \mathbf{u}_b g_{bi_2}^{(1)} &= \psi_1(l_{i_2}) - \sum_{b=\alpha}^{k_1-1} \mathbf{u}_b g_{bi_2}^{(1)}, \\
&\vdots \\
\sum_{b=0}^{\alpha-1} \mathbf{u}_b g_{bi_\alpha}^{(1)} &= \psi_1(l_{i_\alpha}) - \sum_{b=\alpha}^{k_1-1} \mathbf{u}_b g_{bi_\alpha}^{(1)}.
\end{aligned}$$

By arguments identical to those made in Claim 13 it can be seen that this system of equations has $q_1^{k_1-\alpha}$ solutions. Applying the same argument to the other cyclic codes we conclude that the vector $[l_{i_1}, l_{i_2}, \cdots, l_{i_\alpha}]$ appears $q_1^{k_1-\alpha} q_2^{k_2-\alpha} \cdots q_d^{k_d-\alpha}$ times in $\mathbf{T}_{\mathcal{I}}$ and the result follows.

APPENDIX B. SUPPLEMENT FOR NUMERICALLY STABLE CODED MATRIX COMPUTATIONS VIA CIRCULANT AND ROTATION MATRIX EMBEDDINGS

Examples of4×4 circulant permutation matrices

Example 20. For $m = 4$, the four possible circulation permutation matrices are

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \mathbf{P}^0 = \mathbf{I}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\mathbf{P}^2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \mathbf{P}^3 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Proof of Claim 12

Proof. Note that Algorithm 5 is applied for recovering the corresponding entries of $\mathbf{A}_{i,j}^T\mathbf{x}$ for $i \in [k_A], j \in [\tilde{q}]$ separately. There are $r/(k_A(q-1))$ such entries. The complexity of computing a N-point FFT is $O(N \log N)$ in terms of the required floating point operations (flops). Computing the permutation does not cost any flops and its complexity is negligible as compared to the other steps. Step 1 of Algorithm 5 therefore has complexity $O(k_A\tilde{q}\log\tilde{q})$. In Step 2, we solve the degree $k_A - 1$ polynomial interpolation, $(\tilde{q}-1)$ times. This takes $O((\tilde{q}-1)k_A \log^2 k_A)$ time Pan (2013). Finally, Step 3, requires applying the inverse permutation and the inverse FFT; this requires $O(k_A\tilde{q}\log\tilde{q})$

operations. Therefore, the overall complexity is given by

$$\begin{aligned}
&\frac{r}{k_A(\tilde{q}-1)}\big(O(k_A\tilde{q}\log\tilde{q}) + O((\tilde{q}-1)k_A\log k_A^2)\big)\\
&\approx O(r(\log\tilde{q} + \log^2 k_A)).
\end{aligned}$$

□

Proof of Theorem 5

Proof. The arguments are conceptually similar to the proof of Theorem 4. Suppose that the workers indexed by $i_0, \ldots, i_{k_A-1}$ complete their tasks. The corresponding block columns of $\mathbf{G}^{circ}$ can be extracted to form

$$\tilde{\mathbf{G}} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} \\ \mathbf{P}^{i_0} & \mathbf{P}^{i_1} & \cdots & \mathbf{P}^{i_{k_A-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}^{i_0(k_A-1)} & \mathbf{P}^{i_1(k_A-1)} & \cdots & \mathbf{P}^{i_{k_A-1}(k_A-1)} \end{bmatrix}.$$

As in the proof of Theorem 4 we can equivalently analyze the decoding by considering the system of equations

$$\mathbf{m}\tilde{\mathbf{G}} = \mathbf{c},$$

where $\mathbf{m}, \mathbf{c} \in \mathbb{R}^{1\times k_A\tilde{q}}$ are row-vectors such that

$$\begin{aligned}
\mathbf{m} &= [\mathbf{m}_0, \cdots, \mathbf{m}_{k_A-1}]\\
&= [\mathbf{m}_{\langle 0,0\rangle}, \cdots, \mathbf{m}_{\langle 0,\tilde{q}-1\rangle}, \cdots, \mathbf{m}_{\langle k_A-1,0\rangle}, \cdots, \mathbf{m}_{\langle k_A-1,\tilde{q}-1\rangle}], \text{ and}\\
\mathbf{c} &= [\mathbf{c}_{i_0}, \cdots, \mathbf{c}_{i_{k_A-1}}]\\
&= [\mathbf{c}_{\langle i_0,0\rangle}, \cdots, \mathbf{c}_{\langle i_0,\tilde{q}-1\rangle}, \cdots, \mathbf{c}_{\langle i_{k_A-1},0\rangle}, \cdots, \mathbf{c}_{\langle i_{k_A-1},\tilde{q}-1\rangle}].
\end{aligned}$$

Note that not all variables in $\mathbf{m}$ are independent owing to (5.7). Let $\mathbf{m}^{\mathcal{F}}$ and $\mathbf{c}^{\mathcal{F}}$ denote the $\tilde{q}$-point "block-Fourier" transforms of these vectors, i.e,

$$\mathbf{m}^{\mathcal{F}} = \mathbf{m}\begin{bmatrix}\mathbf{W} & & \\ & \ddots & \\ & & \mathbf{W}\end{bmatrix} \text{ and}$$

$$\mathbf{c}^{\mathcal{F}} = \mathbf{c}\begin{bmatrix}\mathbf{W} & & \\ & \ddots & \\ & & \mathbf{W}\end{bmatrix},$$

where $\mathbf{W}$ is the $\tilde{q}$-point DFT matrix. Let $\tilde{\mathbf{G}}_{k,l} = \mathbf{P}^{i_l k}$ denote the (k,l)-th block of $\tilde{\mathbf{G}}$. Using the fact that $\mathbf{P}$ can be diagonalized by the DFT matrix $\mathbf{W}$, we have

$$\tilde{\mathbf{G}}_{k,l} = \mathbf{W}\mathrm{diag}(1, \omega_{\tilde{q}}^{i_l k}, \omega_{\tilde{q}}^{2 i_l k}, \ldots, \omega_{\tilde{q}}^{(\tilde{q}-1)i_l k})\mathbf{W}^*.$$

Let $\tilde{\mathbf{G}}^{\mathcal{F}}_{k,l} = \mathrm{diag}(1, \omega_{\tilde{q}}^{i_l k}, \omega_{\tilde{q}}^{2 i_l k}, \ldots, \omega_{\tilde{q}}^{(\tilde{q}-1)i_l k})$, and $\tilde{\mathbf{G}}^{\mathcal{F}}$ represent the $k_A \times k_A$ block matrix with $\tilde{\mathbf{G}}^{\mathcal{F}}_{k,l}$ for $k, l = 0, \ldots, k_A - 1$ as its blocks. Therefore, the system of equations

$$\mathbf{m}\tilde{\mathbf{G}} = \mathbf{c},$$

can be further expressed as

$$\mathbf{m}\begin{bmatrix}\mathbf{W} & & \\ & \ddots & \\ & & \mathbf{W}\end{bmatrix}\begin{bmatrix}\mathbf{W}^* & & \\ & \ddots & \\ & & \mathbf{W}^*\end{bmatrix}\tilde{\mathbf{G}}\begin{bmatrix}\mathbf{W} & & \\ & \ddots & \\ & & \mathbf{W}\end{bmatrix} = \mathbf{c}\begin{bmatrix}\mathbf{W} & & \\ & \ddots & \\ & & \mathbf{W}\end{bmatrix},$$

$$\implies [\mathbf{m}^{\mathcal{F}}_0, \cdots, \mathbf{m}^{\mathcal{F}}_{k_A-1}]\tilde{\mathbf{G}}^{\mathcal{F}} = [\mathbf{c}^{\mathcal{F}}_{i_0}, \cdots, \mathbf{c}^{\mathcal{F}}_{i_{k_A-1}}]$$

upon right multiplication by the matrix $\begin{bmatrix}\mathbf{W} & & \\ & \ddots & \\ & & \mathbf{W}\end{bmatrix}$. Next, we note that as each block within $\tilde{\mathbf{G}}^{\mathcal{F}}$ has a diagonal structure, we can rewrite the system of equations as a block diagonal matrix upon applying an appropriate permutation (*cf.* Claim 14 in Appendix B). Thus, we can rewrite it as

$$[\mathbf{m}_0^{\mathcal{F},\pi},\cdots,\mathbf{m}_{\tilde{q}-1}^{\mathcal{F},\pi}]\tilde{\mathbf{G}}_d^{\mathcal{F}} = [\mathbf{c}_0^{\mathcal{F},\pi},\cdots,\mathbf{c}_{\tilde{q}-1}^{\mathcal{F},\pi}], \tag{B.1}$$

where the permutation π is such that $\mathbf{m}_j^{\mathcal{F},\pi} = [\mathbf{m}_{0,j}^{\mathcal{F}}\ \mathbf{m}_{1,j}^{\mathcal{F}}\ \ldots\ \mathbf{m}_{k_A-1,j}^{\mathcal{F}}]$ and likewise $\mathbf{c}_j^{\mathcal{F},\pi} = [\mathbf{c}_{i_0,j}^{\mathcal{F}}\ \mathbf{c}_{i_1,j}^{\mathcal{F}}\ \ldots\ \mathbf{c}_{i_{k_A-1},j}^{\mathcal{F}}]$. Furthermore, $\tilde{\mathbf{G}}_d^{\mathcal{F}}$ is a block-diagonal matrix where each block is of size $k_A \times k_A$. Now, according to (5.7), we have $\mathbf{m}_{i,0}^{\mathcal{F}} = \sum_{j=0}^{\tilde{q}-1}\mathbf{m}_{i,j} = 0$ for $i = 0,\ldots,k_A-1$, which implies that $\mathbf{m}_0^{\mathcal{F},\pi}$ is a $1\times k_A$ zero row-vector and thus $\mathbf{c}_0^{\mathcal{F},\pi}$ is too.

In what follows, we show that each of other diagonal blocks of $\tilde{\mathbf{G}}_d^{\mathcal{F}}$ is non-singular. This means that $[\mathbf{m}_0^{\mathcal{F}},\cdots,\mathbf{m}_{k_A-1}^{\mathcal{F}}]$ and consequently $\mathbf{m}$ can be determined by solving the system of equations in (B.1). Towards this end, we note that the k-th diagonal block ($1\le k\le \tilde{q}-1$) of $\tilde{\mathbf{G}}_d^{\mathcal{F}}$, denoted by $\tilde{\mathbf{G}}_d^{\mathcal{F}}[k]$ can be expressed as follows.

$$\tilde{\mathbf{G}}_d^{\mathcal{F}}[k] = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \omega_{\tilde{q}}^{i_0 k} & \omega_{\tilde{q}}^{i_1 k} & \cdots & \omega_{\tilde{q}}^{i_{k_A-1}k} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_{\tilde{q}}^{(k_A-1)i_0 k} & \omega_{\tilde{q}}^{(k_A-1)i_1 k} & \cdots & \omega_{\tilde{q}}^{(k_A-1)i_{k_A-1}k} \end{bmatrix}. \tag{B.2}$$

The above matrix is a *complex* Vandermonde matrix with parameters $\omega_{\tilde{q}}^{i_0 k},\ldots,\omega_{\tilde{q}}^{i_{k_A-1}k}$. Thus, as long these parameters are distinct, $\tilde{\mathbf{G}}_d^{\mathcal{F}}[k]$ will be non-singular. Note that we need the property to hold for $k = 1,\ldots,\tilde{q}-1$. This condition can be expressed as

$$(i_\alpha - i_\beta)k \not\equiv 0 \pmod{\tilde{q}},$$

for $i_\alpha, i_\beta \in \{0,\ldots,n-1\}$ and $1\le k\le \tilde{q}-1$. A necessary and sufficient condition for this to hold is that $\tilde{q}$ is prime. An application of Theorem 3 shows that $\kappa(\tilde{\mathbf{G}}_d^{\mathcal{F}}[k]) \le O(\tilde{q}^{\tilde{q}-k_A+6})$ for all k. As decoding $\mathbf{m}$ is equivalent to solving systems of equations specified by $\tilde{\mathbf{G}}_d^{\mathcal{F}}[k]$ for $1\le k\le\tilde{q}-1$, the worst case condition number is at most $O(\tilde{q}^{\tilde{q}-k_A+6})$. □

Vandermonde Matrix condition number analysis

Let $\mathbf{V}$ be a $m \times m$ Vandermonde matrix with parameters $s_0, s_1, \dots s_{m-1}$. We are interested in upper bounding $\kappa(\mathbf{V})$. Let $s_+ = \max_{i=0}^{m-1} |s_i|$. Then, it is known that $||\mathbf{V}|| \leq m \max(1, s_+^{n-1})$ Pan (2016). Finding an upper bound on $||\mathbf{V}^{-1}||$ is more complicated and we discuss this in detail below. Towards this end we need the definition of a Cauchy matrix.

Definition 12. A $m \times m$ Cauchy matrix is specified by parameters $\mathbf{s} = [s_0 \ s_1 \ \dots \ s_{m-1}]$ and $\mathbf{t} = [t_0 \ t_1 \ \dots \ t_{m-1}]$, such that its (i,j)-th entry

$$\mathbf{C}_{\mathbf{s},\mathbf{t}}(i,j) = \left(\frac{1}{s_i - t_j}\right) \text{ for } i \in [m], j \in [m].$$

In what follows, we establish an upper bound on the condition number of Vandermonde matrices with parameters on the unit circle.

Theorem 8. Consider a $m \times m$ Vandermonde matrix $\mathbf{V}$ where $m < q$ (q is odd) with distinct parameters $\{s_0, s_1, \dots, s_{m-1}\} \subset \{1, \omega_q, \omega_q^2, \dots, \omega_q^{q-1}\}$. Then,

$$\kappa(\mathbf{V}) \leq O(q^{q-m+6}).$$

Proof. Recall that $\omega_q = e^{\mathrm{i}\frac{2\pi}{\mathrm{q}}}$ and $\omega_m = e^{\mathrm{i}\frac{2\pi}{\mathrm{m}}}$ and define $t_j = f\omega_m^j, j = 0, \dots, m-1$ where f is a complex number with $|f| = 1$. We let $\mathbf{C}_{\mathbf{s},f}$ denote the Cauchy matrix with parameters $\{s_0, \dots, s_{m-1}\}$ and $\{t_0, \dots, t_{m-1}\}$. Let $\mathbf{W}$ be the m-point DFT matrix. The work of Pan (2016) shows that

$$\mathbf{V}^{-1} = \operatorname{diag}(f^{m-1-j})_{j=0}^{m-1} \mathbf{W}^* \operatorname{diag}(\omega_m^{-j})_{j=0}^{m-1} \mathbf{C}_{\mathbf{s},f}^{-1} \operatorname{diag}\left(\frac{1}{s_j^m - f^m}\right)_{j=0}^{m-1}.$$

It can be seen that the matrix $\operatorname{diag}(f^{m-1-j})_{j=0}^{m-1} \mathbf{W}^* \operatorname{diag}(\omega_m^{-j})_{j=0}^{m-1}$ is unitary. Therefore,

$$\begin{aligned}
||\mathbf{V}^{-1}|| &= ||\mathbf{C}_{\mathbf{s},f}^{-1} \operatorname{diag}\left(\frac{1}{s_j^m - f^m}\right)_{j=0}^{m-1} || \\
&\leq ||\mathbf{C}_{\mathbf{s},f}^{-1}|| \times \left(\frac{1}{\min_{i=0}^{m-1} |s_i^m - f^m|}\right) \\
&\leq m \times (\max_{i',j'} |\mathbf{C}_{\mathbf{s},f}^{-1}(i',j')|) \times \left(\frac{1}{\min_{i=0}^{m-1} |s_i^m - f^m|}\right), \qquad \text{(B.3)}
\end{aligned}$$

where the first inequality holds as the norm of a product of matrices is upper bounded by the products of the individual norms and second inequality holds since for any $\mathbf{M}$, we have $||\mathbf{M}|| \leq ||\mathbf{M}||_F$.

In what follows, we upper bound the RHS of (B.3). Let $s(x)$ denote a function of x so that $s(x) = \Pi_{i=0}^{m-1}(x - s_i)$. The (i', j')-the entry of $\mathbf{C}_{\mathbf{s},f}^{-1}$ can be expressed as Pan (2016)

$$\begin{aligned}
\mathbf{C}_{\mathbf{s},f}^{-1}(i', j') &= (-1)^m s(t_{j'})(s_{i'}^m - f^m)/(s_{i'} - t_{j'}), \text{ so that} \\
|\mathbf{C}_{\mathbf{s},f}^{-1}(i', j')| &= |s(t_{j'})||s_{i'}^m - f^m|/|s_{i'} - t_{j'}| \\
&\leq |s(t_{j'})|(|s_{i'}^m| + |f^m|)/|s_{i'} - t_{j'}| \\
&= 2|s(t_{j'})|/|s_{i'} - t_{j'}| \ \ (\text{since } |s_{i'}| = |f| = 1).
\end{aligned}$$

Let $\mathcal{M} = \{1, \omega_q, \omega_q^2, \ldots, \omega_q^{q-1}\} \setminus \{s_0, s_1, \ldots, s_{m-1}\}$ denote the q-th roots of unity that are *not* parameters of $\mathbf{V}$. Note that

$$\begin{aligned}
s(t_{j'}) &= \Pi_{i=0}^{m-1}(t_{j'} - s_i) \\
&= \left.\frac{x^q - 1}{\Pi_{\alpha_j \in \mathcal{M}}(x - \alpha_j)}\right|_{x = t_{j'}}, \text{ so that} \\
|s(t_{j'})| &= \frac{|t_{j'}^q - 1|}{\Pi_{\alpha_j \in \mathcal{M}}|t_{j'} - \alpha_j|} \\
&\leq \frac{2}{\Pi_{\alpha_j \in \mathcal{M}}|t_{j'} - \alpha_j|} \ (\text{since } |t_{j'}| = 1 \text{ and by the triangle inequality}).
\end{aligned}$$

Thus, we can conclude that

$$|\mathbf{C}_{\mathbf{s},f}^{-1}(i', j')| \leq 4 \max_{i', j'} \frac{1}{\Pi_{\alpha_j \in \mathcal{M}}|(t_{j'} - \alpha_j)|} \frac{1}{|s_{i'} - t_{j'}|} \tag{B.4}$$

$$= 4\left(\frac{1}{\min_{i', j'} \Pi_{\alpha_j \in \mathcal{M}}|(t_{j'} - \alpha_j)|} \frac{1}{|s_{i'} - t_{j'}|}\right). \tag{B.5}$$

Note that in the expression above, $s_{i'}$ is a parameter of $\mathbf{V}$ while the α_j's are the points within $\Omega_q = \{1, \omega_q, \omega_q^2, \ldots, \omega_q^{q-1}\}$ that are "not" parameters of $\mathbf{V}$. We choose $f = e^{\mathrm{i}\frac{\pi}{m}}$ so that $t_{j'} = f\omega_m^{j'} = e^{\mathrm{i}\pi/m}\omega_m^{j'}$. Next, we determine an upper bound on the RHS of (B.5). Towards this end, we note that the distance between two points on the unit circle can be expressed as $2\sin(\theta/2)$ if θ is the induced angle between them. Furthermore, we have $2\sin(\theta/2) \geq 2\theta/\pi$ as long as $\theta \leq \pi$.

It can be seen that the closest point to $t_{j'}$ that lies within Ω_q has an induced angle

$$\frac{2\pi\ell}{q} - \frac{2\pi(j'+\frac{1}{2})}{m} \geq \frac{2\pi}{qm}\frac{1}{2} \geq \frac{\pi}{q^2}.$$

Therefore, the corresponding distance is lower bounded by $2/q^2$. Similarly, the next closest distance is lower bounded by $2/q$, followed by $2(2/q), 3(2/q), \ldots, (q-m-1)(2/q)$. Let $d = q - m$, Then,

$$\begin{aligned}
&\left(\Pi_{\alpha_j \in \mathcal{M}} |(t_{j'} - \alpha_j)|\right) \min_{i',j'} |s_{i'} - t_{j'}| \\
&\geq 2/q^2 \times 2/q \times 4/q \times \cdots \times 2(d-1)/q \times 2/q^2 \\
&= 2^{d+1}(d-1)!\frac{1}{q^{d+3}}.
\end{aligned}$$

Therefore,

$$|\mathbf{C}_{\mathbf{s},f}^{-1}(i',j')| \leq \frac{q^{d+3}}{C_d}$$

where $C_d = 2^{d-1}(d-1)!$ is a constant. Let the i-th parameter $s_i = e^{\mathrm{i}2\pi\ell/\mathrm{q}}$. Then,

$$\begin{aligned}
|s_i^m - f^m| &= |e^{\mathrm{i}2\pi\ell\mathrm{m}/\mathrm{q}} + 1| \\
&= 2|\cos(\pi\ell m/q)|.
\end{aligned}$$

The term ℓm can be expressed as $\ell m = \beta q + \eta$ for integers β and η such that $0 \leq \eta \leq q-1$. Now note that $\eta \neq q/2$ since by assumption q is odd. Thus, $|\cos(\pi\ell m/q)|$ takes its smallest value when $\eta = (q+1)/2$ or $(q-1)/2$. In this case

$$\begin{aligned}
|\cos(\pi\ell m/q)| &= \left|\cos\left(\beta\pi + \pi\frac{q+1}{2q}\right)\right| \\
&\geq \left|\sin\left(\frac{\pi}{2q}\right)\right| \\
&\geq \frac{1}{q}.
\end{aligned}$$

Thus, we can upper bound the RHS of (B.3) and obtain

$$\begin{aligned}
||\mathbf{V}^{-1}|| &\leq m\frac{q^{d+3}}{C_d}q \\
&\leq \frac{q^{d+5}}{C_d} \text{ (since } m < q).
\end{aligned}$$

Finally, using the fact that $||V|| \leq m < q$. we obtain

$$\kappa(\mathbf{V}) \leq \frac{q^{d+6}}{C_d}.$$

□

Auxiliary Claims

Definition 13. *Permutation Equivalence.* We say that a matrix $\mathbf{M}$ is *permutation equivalent* to $\mathbf{M}^{\pi}$ if $\mathbf{M}^{\pi}$ can be obtained by permuting the rows and columns of $\mathbf{M}$. We denote this by $\mathbf{M} \asymp \mathbf{M}^{\pi}$.

Claim 14. Let $\mathbf{M}$ be a $l_1 q \times l_2 q$ matrix consisting of blocks of size $q \times q$ denoted by $\mathbf{M}_{i,j}$ for $i \in [l_1], j \in [l_2]$. Each $\mathbf{M}_{i,j}$ is a diagonal matrix. Then, the rows and columns of $\mathbf{M}$ can be permuted to obtain $\mathbf{M}^{\pi}$ which is a block diagonal matrix where each block matrix is of size $l_1 \times l_2$ and there are q of them.

Proof. For an integer a, let $(a)_q$ denote a mod q. In what follows, we establish two permutations

$$\pi_{l_1}(i) = l_1(i)_q + \lfloor i/q \rfloor, 0 \leq i < l_1 q, \text{ and}$$
$$\pi_{l_2}(j) = l_2(j)_q + \lfloor j/q \rfloor, 0 \leq j < l_2 q$$

and show that applying row-permutation π_{l_1} and column-permutation π_{l_2} to $\mathbf{M}$ will result in a block diagonal matrix $\mathbf{M}^{\pi}$.

We observe that (i,j)-th entry in $\mathbf{M}$ is the $((i)_q,(j)_q)$-th entry in $\mathbf{M}_{\lfloor i/q \rfloor, \lfloor j/q \rfloor}$. Under the applied permutations the (i,j)-th entry in $\mathbf{M}$ is mapped to $(l_1(i)_q + \lfloor i/q \rfloor, l_2(j)_q + \lfloor j/q \rfloor)$-entry in $\mathbf{M}^{\pi}$. Recall that $\mathbf{M}_{\lfloor i/q \rfloor, \lfloor j/q \rfloor}$ is a diagonal matrix which implies that for $(i)_q \neq (j)_q$, the $(l_1(i)_q + \lfloor i/q \rfloor, l_2(j)_q + \lfloor j/q \rfloor)$ entry in $\mathbf{M}^{\pi}$ is 0. Therefore $\mathbf{M}^{\pi}$ is a block diagonal matrix with q blocks of size $l_1 \times l_2$. □

Example 21. Let $l_1 = 2, l_2 = 3, q = 2$. Consider a 4×6 matrix $\mathbf{M}$ which consists of diagonal matrices $\mathbf{M}_{i,j}$ of size 2×2. For $0 \leq i \leq 1, 0 \leq j \leq 2$

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_{0,0} & \mathbf{M}_{0,1} & \mathbf{M}_{0,2} \\ \mathbf{M}_{1,0} & \mathbf{M}_{1,1} & \mathbf{M}_{1,2} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & \omega_q & 0 & \omega_q^2 & 0 \\ 0 & 1 & 0 & \omega_q^{-1} & 0 & \omega_q^{-2} \end{bmatrix}.$$

We use row permutation $\pi_{\text{row}} = (0, 2, 1, 3)$, which means $0, 1, 2, 3$-th row of $\mathbf{M}$ permutes to $0, 2, 1, 3$-th row. Similarly, the column permutation is $\pi_{\text{col}} = (0, 3, 1, 4, 2, 5)$. Thus, $\mathbf{M}^{\pi}$ becomes

$$\mathbf{M}^{\pi} = \begin{bmatrix} 1 & 1 & 1 & & & \\ 1 & \omega_q & \omega_q^2 & & & \\ & & & 1 & 1 & 1 \\ & & & 1 & \omega_q^{-1} & \omega_q^{-2} \end{bmatrix}.$$

Claim 15. (i) Let $a_0(z) = \sum_{j=0}^{\ell_a - 1} a_{j0} z^j$, $a_1(z) = \sum_{j=0}^{\ell_a - 1} a_{j1} z^{-j}$ and $b_0(z) = \sum_{j=0}^{\ell_b - 1} b_{j0} z^{j\ell_a}$, $b_1(z) = \sum_{j=0}^{\ell_b - 1} b_{j1} z^{-j\ell_a}$. Then, $a_{k_1}(z) b_{k_2}(z)$ for $k_1, k_2 = 0, 1$ are polynomials that can be recovered from $\ell_a \ell_b$ distinct evaluation points in $\mathbb{C}$.

Let $\mathbf{D}(z^j) = \text{diag}([z^j \; z^{-j}])$ and let

$$\mathbf{X}(z) = \begin{bmatrix} \mathbf{I}_2 \\ \mathbf{D}(z) \\ \vdots \\ \mathbf{D}(z^{\ell_a - 1}) \end{bmatrix} \otimes \begin{bmatrix} \mathbf{I}_2 \\ \mathbf{D}(z^{\ell_a}) \\ \vdots \\ \mathbf{D}(z^{\ell_a(\ell_b - 1)}) \end{bmatrix}.$$

Then, if z_i's are distinct points in $\mathbb{C}$, the matrix

$$[\mathbf{X}(z_1) | \mathbf{X}(z_2) | \dots | \mathbf{X}(z_{\ell_a \ell_b})],$$

is nonsingular.

(ii) The matrix $[\mathbf{X}_{i_0}|\mathbf{X}_{i_1}|\dots|\mathbf{X}_{i_{\tau-1}}]$ (defined in the proof of Theorem 6) is permutation equivalent to a block-diagonal matrix with four blocks each of size $\tau \times \tau$. Each of these blocks is a Vandermonde matrix with parameters from the set $\{1, \omega_q, \omega_q^2, \dots, \omega_q^{q-1}\}$.

Proof. First we show that $a_{k_1}(z)b_{k_2}(z)$ for $k_1, k_2 = 0, 1$ are polynomials that can be recovered from $\ell_a\ell_b$ distinct evaluation points in $\mathbb{C}$. Towards this end, these four polynomials can be written as

$$\begin{aligned}
a_0(z)b_0(z) &= \sum_{i=0}^{\ell_a-1}\sum_{j=0}^{\ell_b-1} a_{i0}b_{j0}z^{i+j\ell_a},\\
a_0(z)b_1(z) &= \sum_{i=0}^{\ell_a-1}\sum_{j=0}^{\ell_b-1} a_{i0}b_{j1}z^{i-j\ell_a},\\
a_1(z)b_0(z) &= \sum_{i=0}^{\ell_a-1}\sum_{j=0}^{\ell_b-1} a_{i1}b_{j0}z^{-i+j\ell_a}, \text{ and}\\
a_1(z)b_1(z) &= \sum_{i=0}^{\ell_a-1}\sum_{j=0}^{\ell_b-1} a_{i1}b_{j1}z^{-i-j\ell_a}.
\end{aligned}$$

Upon inspection, it can be seen that each of the polynomials above has $\ell_a\ell_b$ consecutive powers of z. Therefore, each of these can be interpolated from $\ell_a\ell_b$ non-zero distinct evaluation points in $\mathbb{C}$.

The second part of the claim follows from the above discussion. To see this we note that

$$[a_0(z)\ a_1(z)] = [a_{00}\ a_{01}\ a_{10}\ a_{11}\ \dots a_{(\ell_a-1)0}\ a_{(\ell_a-1)1}] \begin{bmatrix} \mathbf{I}_2 \\ \mathbf{D}(z) \\ \vdots \\ \mathbf{D}(z^{\ell_a-1}) \end{bmatrix} \text{ and}$$

$$[b_0(z)\ b_1(z)] = [b_{00}\ b_{01}\ b_{10}\ b_{11}\ \dots b_{(\ell_b-1)0}\ b_{(\ell_b-1)1}] \begin{bmatrix} \mathbf{I}_2 \\ \mathbf{D}(z^{\ell_a}) \\ \vdots \\ \mathbf{D}(z^{\ell_a(\ell_b-1)}) \end{bmatrix}.$$

Furthermore, the four product polynomials under consideration can be expressed as

$$\begin{aligned}
&[a_0(z)\ a_1(z)] \otimes [b_0(z)\ b_1(z)]\\
&= \left([a_{00}\ a_{01}\ a_{10}\ a_{11}\ \dots a_{(\ell_a-1)0}\ a_{(\ell_a-1)1}] \otimes [b_{00}\ b_{01}\ b_{10}\ b_{11}\ \dots b_{(\ell_b-1)0}\ b_{(\ell_b-1)1}]\right) \mathbf{X}(z).
\end{aligned}$$

We have previously shown that all polynomials in $[a_0(z)\ a_1(z)] \otimes [b_0(z)\ b_1(z)]$ can be interpolated by obtaining their values on $\ell_a\ell_b$ non-zero distinct evaluation points. This implies that we can equivalently obtain

$$\big([a_{00}\ a_{01}\ a_{10}\ a_{11}\ \ \ldots a_{(\ell_a-1)0}\ a_{(\ell_a-1)1}] \otimes [b_{00}\ b_{01}\ b_{10}\ b_{11}\ \ \ldots b_{(\ell_b-1)0}\ b_{(\ell_b-1)1}]\big)$$

which means that $[\mathbf{X}(z_1)|\mathbf{X}(z_2)|\ldots|\mathbf{X}(z_{\ell_a\ell_b})]$ is non-singular. This proves the statement in part (i).

The proof of the statement in (ii) is essentially an exercise in showing the permutation equivalence of several matrices by using Claim 14 and the permutation equivalence properties of Kronecker products. For convenience, we define

$$\mathbf{X}_{l,A} = \begin{bmatrix} \mathbf{I} \\ \Lambda^{l} \\ \vdots \\ \Lambda^{l(k_A-1)} \end{bmatrix}, \text{ and}$$

$$\mathbf{X}_{l,B} = \begin{bmatrix} \mathbf{I} \\ \Lambda^{lk_A} \\ \vdots \\ \Lambda^{lk_A(k_B-1)} \end{bmatrix}$$

so that $\mathbf{X}_l = \mathbf{X}_{l,A} \otimes \mathbf{X}_{l,B}$.

Recall that we are analyzing the matrix $\mathbf{X} = [\mathbf{X}_{i_0}|\mathbf{X}_{i_1}|\ldots|\mathbf{X}_{i_{\tau-1}}]$. An application of Claim 14 shows that

$$\mathbf{X}_{l,A} \asymp \mathbf{X}^P_{l,A} = \begin{bmatrix} \mathbf{V}_{l,A,1} & \\ & \mathbf{V}_{l,A,2} \end{bmatrix}, \text{ and } \mathbf{X}_{l,B} \asymp \mathbf{X}^P_{l,B} = \begin{bmatrix} \mathbf{V}_{l,B,1} & \\ & \mathbf{V}_{l,B,2} \end{bmatrix},$$

where $\mathbf{V}_{l,A,1} = [1, \omega_q^{l}, \cdots, \omega_q^{l(k_A-1)}]^T$, $\mathbf{V}_{l,A,2} = [1, \omega_q^{-l}, \cdots, \omega_q^{-l(k_A-1)}]^T$, $\mathbf{V}_{l,B,1} = [1, \omega_q^{lk_A}, \cdots, \omega_q^{lk_A(k_B-1)}]^T$, $\mathbf{V}_{l,B,2} = [1, \omega_q^{-lk_A}, \cdots, \omega_q^{-lk_A(k_B-1)}]^T$. Then we conclude that $\mathbf{X} \asymp \mathbf{X}^P = [\mathbf{X}^P_{i_0}|\mathbf{X}^P_{i_1}|\cdots|\mathbf{X}^P_{i_{\tau-1}}]$, where $\mathbf{X}_l^P = \mathbf{X}_{l,A}^P \otimes \mathbf{X}_{l,B}^P$. Next we show that

$$\mathbf{X}_l^P = \mathbf{X}_{l,A}^P \otimes \mathbf{X}_{l,B}^P \asymp \mathbf{X}_l^{P,\pi} = \begin{bmatrix} \mathbf{V}_{l,A,1} \otimes \mathbf{V}_{l,B,1} & & & \\ & \mathbf{V}_{l,A,2} \otimes \mathbf{V}_{l,B,1} & & \\ & & \mathbf{V}_{l,A,1} \otimes \mathbf{V}_{l,B,2} & \\ & & & \mathbf{V}_{l,A,2} \otimes \mathbf{V}_{l,B,2} \end{bmatrix}.$$

By the definition of Kronecker product, we have

$$\mathbf{X}_{l,A}^P \otimes \mathbf{X}_{l,B}^P = \begin{bmatrix} \mathbf{V}_{l,A,1} \otimes \mathbf{X}_{l,B}^P & \\ & \mathbf{V}_{l,A,2} \otimes \mathbf{X}_{l,B}^P \end{bmatrix}.$$

Note that $\mathbf{V}_{l,A,i} \otimes \mathbf{V}_{l,B,j} \asymp \mathbf{V}_{l,B,j} \otimes \mathbf{V}_{l,A,i}$, then

$$\begin{aligned} & \mathbf{V}_{l,A,i} \otimes \mathbf{X}_{l,B}^P \\ =& \mathbf{V}_{l,A,i} \otimes \begin{bmatrix} \mathbf{V}_{l,B,1} & \\ & \mathbf{V}_{l,B,2} \end{bmatrix} \\ \asymp & \begin{bmatrix} \mathbf{V}_{l,B,1} & \\ & \mathbf{V}_{l,B,2} \end{bmatrix} \otimes \mathbf{V}_{l,A,i} \\ = & \begin{bmatrix} \mathbf{V}_{l,B,1} \otimes \mathbf{V}_{l,A,i} & \\ & \mathbf{V}_{l,B,2} \otimes \mathbf{V}_{l,A,i} \end{bmatrix} \\ \asymp & \begin{bmatrix} \mathbf{V}_{l,A,i} \otimes \mathbf{V}_{l,B,1} & \\ & \mathbf{V}_{l,A,i} \otimes \mathbf{V}_{l,B,2} \end{bmatrix}. \end{aligned}$$

Thus, we can conclude that $\mathbf{X}_l^P \asymp \mathbf{X}_l^{P,\pi}$. In addition, we have

$$\begin{aligned} \mathbf{V}_{l,A,1} \otimes \mathbf{V}_{l,B,1} &= [1, \omega_q^l, \cdots, \omega_q^{l(k_A k_B - 2)}, \omega_q^{l(k_A k_B - 1)}]^T, \\ \mathbf{V}_{l,A,2} \otimes \mathbf{V}_{l,B,1} &= [\omega_q^{-l(k_A-1)}, \omega_q^{-l(k_A-2)}, \cdots, \omega_q^{-l}, 1, \omega_q^l, \cdots, \omega_q^{l(k_A(k_B-1)-1)}, \omega_q^{lk_A(k_B-1)}]^T, \\ \mathbf{V}_{l,A,1} \otimes \mathbf{V}_{l,B,2} &= [\omega_q^{-lk_A(k_B-1)}, \omega_q^{-l(k_A(k_B-1)-1)}, \cdots, \omega_q^{-l}, 1, \omega_q^l, \cdots, \omega_q^{l(k_A-2)}, \omega_q^{l(k_A-1)}]^T, \text{ and} \\ \mathbf{V}_{l,A,2} \otimes \mathbf{V}_{l,B,2} &= [\omega_q^{-l(k_A k_B - 1)}, \omega_q^{-l(k_A k_B - 2)}, \cdots, \omega_q^{-l}, 1]^T. \end{aligned}$$

Finally applying Claim 14 again we obtain the required result. □

B.0.1 Proof of Lemma 4

Proof. The proof is essentially an exercise in the definition of Kronecker products.

$$\begin{aligned}
\begin{bmatrix} \Psi_{0,0}\mathbf{M}_1 + \Psi_{0,1}\mathbf{M}_2 \\ \Psi_{1,0}\mathbf{M}_1 + \Psi_{1,1}\mathbf{M}_2 \end{bmatrix} &= \begin{bmatrix} \Psi_{0,0}\mathbf{I}_\zeta\mathbf{M}_1 + \Psi_{0,1}\mathbf{I}_\zeta\mathbf{M}_2 \\ \Psi_{1,0}\mathbf{I}_\zeta\mathbf{M}_1 + \Psi_{1,1}\mathbf{I}_\zeta\mathbf{M}_2 \end{bmatrix} \\
&= \begin{bmatrix} \Psi_{0,0}\mathbf{I}_\zeta & \Psi_{0,1}\mathbf{I}_\zeta \\ \Psi_{1,0}\mathbf{I}_\zeta & \Psi_{1,1}\mathbf{I}_\zeta \end{bmatrix} \begin{bmatrix} \mathbf{M}_1 \\ \mathbf{M}_2 \end{bmatrix} \\
&= \begin{bmatrix} \Psi_{0,0} & \Psi_{0,1} \\ \Psi_{1,0} & \Psi_{1,1} \end{bmatrix} \otimes \mathbf{I}_\zeta \begin{bmatrix} \mathbf{M}_1 \\ \mathbf{M}_2 \end{bmatrix}
\end{aligned}$$

□

Claim 16. Let $\tau_{dif} = 2k_Ak_Bp - 2(k_Ak_B + pk_A + pk_B) + k_A + k_B + 2p$ and $p > 1$. $\tau_{dif} < 0$ only if $k_A = 1$ or $k_B = 1$.

Proof. We fix k_B and p and consider τ_{dif} is a function of k_A. Then

$$\tau'_{dif}(k_A) = 2k_Bp - 2k_B - 2p + 1 > 0,$$

which means $\tau_{dif}(k_A)$ is a strictly increasing function. We consider the following three cases,

- $k_A = 1$. In this case, $\tau_{dif}(1) = 1 - k_B$. Then $\tau_{dif} \leq 0$.
- $k_A = 2$. In this case $\tau_{dif}(2) = 2k_Bp - 3k_B - 2p + 2$. It is not clear to evaluate the value of $\tau_{dif}(2)$ directly, thus we consider $\tau_{dif}(2)$ be a function of k_B. $\tau_{dif}(2)'(k_B) = 2p - 3 > 0$, which means $\tau_{dif}(2)'(k_B)$ is a strictly increasing function since in this proof we assume $p > 2$. When $k_B = 2$, $\tau'_{dif}(2)(2) = 2p - 4 \geq 0$. Then we conclude $\tau_{dif}(2) \geq 0$.
- $k_A > 2$. It is clear that $\tau_{dif}(k_A) > 0$ since $\tau_{dif}(k_A)$ is strictly increasing and $\tau_{dif}(2) \geq 0$.

The same argument can be applied to k_B. □